Pricing Los Angeles Expressways

Shuyang Li
Max A. Kaplan

May 13, 2014
ORF 418 | Professor Warren Powell

Table of Contents

1. Introduction

In Los Angeles, the Metro created Express Lanes on the I-110 Freeway about a year and a half ago. The Express Lanes are lanes (one or two each direction depending where) parallel to the rest of the freeway where the cost to enter changes dynamically depending on how much time it would save you. These lanes were converted from existing carpool lanes, as the carpool lanes were being underutilized during periods of high traffic (rush hours).

The purpose of these pay lanes were twofold – (1) a source of revenue in part to help underwrite the cost of this massive public works undertaking and (2) to reduce traffic overall by helping to distribute traffic into what were underutilized carpool lanes. The objective function for the government is a combination of revenue (# of cars per unit time in the express lane * price) and overall traffic flow (# of cars per unit time total). However, we will model only the revenue portion.

At any given time, Metro chooses at what price to set the express lanes. Both price and time are continuous but we can discretize. Metro then observes the use of the express lane (and hence the revenue) – and changes the price to learn more (the cost of observing information is zero – the Fastrak sensor in the car measures express lane traffic and we have a distribution for general lane traffic).

This is ultimately a bandit problem-- online learning – especially so in this case since the express lanes started out as a one-month experiment to see how well they worked before continuing their use in subsequent years. Although this is online learning, we will also do it offline too to see how much slower (or if) online finds the optimal price. Also, since this was an experiment to see if the Express Lane program should be spread to other LA freeways, we can treat this as an offline problem because, if the experiment worked so well that it spread across LA, California, or the nation, the revenues from this one freeway for one month could be negligible.

In this project we will first use a lookup table belief model with a(n unknown) logistic truth, and then we will use a local linear approximation of a logistic belief model, with a quadratic revenue curve fitted to the local maximum of the true logistic revenue curve.

2. Mathematical Model

We will start out with a basic offline model. We will be implementing a lookup table belief model for this problem. Then, we will adjust our lookup table belief model for online learning, as our original problem demands.

Then, we will use a logistic model for demand, which we will approximate using a linear belief model in quadratic form for a limited set of alternatives.

2.1 Lookup Table

In this model, the government has constructed the lanes already, and thus its objective is to maximize the revenue from these lanes. We assume the government has two pieces of information: (1) that at P=\$0.00, the traffic is uniformly distributed across the pay and regular lanes, and (2) that demand for the pay lane decreases as one increases the price.

2.1.1 Offline Lookup Table

Objective Function:

$$\zeta = max_P^N \mathbb{E}^\pi[\mu_P^N]$$

where $\mu = R(P) = P \times D(P)$, and $D(P)$ is the demand function (unknown to us) representing the traffic which decides to take the pay lane.

Alternatives:
The alternatives are the prices of the pay lane. We will discretize in \$0.10 increments from \$0 to \$20. This gives us 201 price alternatives.

Observed Values:
In each experiment, we set a price, P, and observe revenue:

$$R^n = R(P) + \epsilon^n,\ \epsilon^n \sim N(0, \sigma^2)$$

Belief Model:
Our first belief model is a lookup table. Our prior is a vector θ_0 with 201 entries, θ_x^0 from x=0 (\$0.00) to x=200 (\$20.00).

We have a covariance function to create the initial covariance matrix:

$$Cov^0\big(R(P), R(P')\big) = \sigma^2 e^{-\alpha|P-P'|}$$

where $\sigma^2 = Var^0(R(P))$. We will adjust σ^2, α to fit public traffic data from the California Department of Transportation.

We will have the true demand function:

$$D(P) = \frac{M}{1 + e^{-\mu_1 + \mu_2 P}}$$

$$M = \frac{\beta}{2}(1 + e^{-\mu_1})$$

β being the peak traffic over the expressway during the sample period.

Prior:
The prior is constructed as follows:

$$\theta_p^0 = \begin{cases} 0.10p\left(-\dfrac{\beta}{10P_{max}}p + \beta\right) & 0 \le p \le 10P_{max} \\ 0 & 10P_{max} < p \le 200 \end{cases}$$

This is constructed as if the government has only two key prior pieces of information: (1) that the demand for the express lane at x=0 (P=$0.00) is half of overall traffic, following from uniform distribution of traffic given non-differentiated pay/regular lanes, and (2) that the demand decreases as price increases, reaching 0 at some maximum price. The government here assumes a linear decline for the demand.

For AM Rush Hour, the government assumes $20.00 for the maximum price; for PM Rush Hour, the government assumes $10.00 for the maximum price; for Graveyard Shift, the government assumes $2.00 for the maximum price.

Updating:
Since we have a covariance matrix, we are using Bayesian Updating with Correlated Beliefs:

$$\theta^{n+1}(p) = \theta^n + \frac{W^{n+1} - \theta_p^n}{\lambda^W + \Sigma_{pp}^n}\Sigma^n e_p$$

$$\Sigma^{n+1}(p) = \Sigma^n - \frac{\Sigma^n e_p (e_p)^T \Sigma^n}{\lambda^W + \Sigma_{pp}^n}$$

Tested Policies:
Knowledge Gradient with Correlated Beliefs
Interval Estimation
Pure Exploitation
Constrained Exploration

Experimental Procedure:
We will test the policies for three distinct time periods of the business day: Morning Rush (7:00 AM - 9:00 AM), Evening Rush (5:00 PM - 8:00 PM), and Graveyard Shift (4:00 AM - 5:00 AM). We will choose the price every 10 minutes and we will simulate each time period over one month (20 business days), giving us a budget of 240 for Morning Rush, 360 for Evening Rush, and 120 for Graveyard Shift.

Each time period has a different demand function. We will obtain the maximum overall traffic volume through traffic data. We will also obtain μ_1, μ_2 from traffic data. There will be higher demand at higher prices during rush hour.

For each simulation, we will start with one of the time periods (splitting it up into three problems). We will set the demand function for that time period (taken from traffic data). We will then loop through N trials, where N is the budget for that time period. For each iteration, the observation is found by

$$R^n = R(P) + \epsilon^n,\ \epsilon^n \sim N(0, \sigma^2)$$

and we will update our belief model. After our budget has been exhausted, we return the value of the function and the final opportunity cost:

$$\Omega = \left| \max_{p \in P} \theta_p^N - \max_{p \in P} \mu_p \right|$$

We will run the simulation T times (100 for UCB, IE, Exploitation, Exploration; 10 for KG due to processing power constraints) and obtained the average final opportunity cost:

$$\overline{\Omega} = \frac{1}{T} \sum_{t=1}^{T} \Omega_t$$

2.1.2 Online Lookup Table

We follow the same principles as offline, except with the online objective function and online policies (Online KGCB, Online UCB, Pure Exploitation, Constrained Exploration). Our simulations will also be online.

Objective Function

$$\zeta = \max_{p \in P} \mathbb{E}^{\pi} \sum_{n=0}^{N} \gamma^n R^n$$

where $\gamma = 1$ in our case, since each day is worth the same to us.

Alternatives:

Refer to Alternatives of **2.1.1 Offline Lookup Table**

Observed Values:
Refer to Observed Values of **2.1.1 Offline Lookup Table**

Belief Model:
Refer to Belief Model of **2.1.1 Offline Lookup Table**

Prior:
Refer to Prior of **2.1.1 Offline Lookup Table**

Updating:
Refer to Updating of **2.1.1 Offline Lookup Table**

Tested Policies:
Knowledge Gradient with Correlated Beliefs
Upper Confidence Bound
Pure Exploitation
Constrained Exploration

Experimental Procedure:
Refer to Experimental Procedure of **2.1.1 Offline Lookup Table.**

However, instead of comparing opportunity costs, we store a value for the cumulative revenue. Following each sample iteration, we add the observed revenue to the cumulative revenue. We will compare policies using cumulative revenue instead of opportunity cost.

2.2 Local Linear Approximation of Logistic Belief Model

In this model, we assume that the government has access to an additional key piece of information: (3) Demand does not decrease linearly with price; at a certain price point, the demand decreases sharply. Thus, the government believes the truth to be logistic. However, the government does not require a full fitted logistic curve, but is satisfied with a linear approximation of the curve within a range around the true maximum. As such, the government here proceeds with a linear belief model for a quadratic revenue function approximation. We will take a quadratic truth as well, best fitting the logistic revenue curve from our Lookup Table model.

2.2.1 Offline Linear Belief Model

Objective Function:

$$\zeta = max_P^N \mathbb{E}^\pi[\mu_P^N]$$

where $\mu = R(P)$, representing the quadratic approximation of the revenue curve about the maximum.

Alternatives:
The alternatives are the set of $x^n = [1 \quad P_n \quad P_n^2]^T$ with P between \$0.00 and \$20.00 corresponding to n between 1 and 201, discretized in \$0.10 increments. Each vector $[1 \quad P_n \quad P_n^2]$ is the set of variables representing the chosen price in the linear model:

$$R(P) = \mu_1 + \mu_2 P + \mu_3 P^2$$

Observed Values:
In each experiment, we set a price, P, and observe revenue:

$$R^n = R(P) + \epsilon^n, \ \epsilon^n \sim N(0, \sigma^2)$$

Belief Model:
In this model, we will use a linear belief model based on a quadratic truth. This quadratic revenue curve is meant to model a section of the true revenue function (based on a logistic demand curve as defined in **2.1.1 Offline Lookup Table**) around its maximum:

$$R(P) = \mu_1 + \mu_2 P + \mu_3 P^2$$

Our true quadratic revenue curve is obtained by finding a set of points on our logistic truth-based revenue curve around its maximum and running a quadratic regression.

Prior:
We construct the prior and the prior covariance matrix Σ^0. First, we assume the government has the same information as in **2.1 Lookup Table**, but with one additional key piece of information: (3) demand does not decrease linearly, and there is a steep decrease at some price point, making the true demand logistic. The government has elected to use a linear belief model (quadratic function) to approximate revenue at and near its maximum, and thus creates a family of 4 likely quadratic curves, each with their individual $\mu_1, \ \mu_2, \ \mu_3$ values. The point estimate of $\mu_1, \ \mu_2, \ \mu_3$ is the mean of the family$\mu_1, \ \mu_2, \ \mu_3$, and the covariance matrix is found as such:

$$Cov^0(\mu_i, \mu_j) = \frac{1}{N-1}\sum_{k=1}^{N}(x_{ki} - \bar{x}_i)(x_{kj} - \bar{x}_j)$$

Updating:
We are using the updating method for linear belief model (section 8.2 of *Optimal Learning*): We have the matrix of alternative choices:

$$X^n = \begin{bmatrix} x_1^1 & \cdots & x_3^1 \\ \vdots & \ddots & \vdots \\ x_1^n & \cdots & x_3^n \end{bmatrix}$$

as well as the matrix:

$$B^n = [(X^n)^T (X^n)]^{-1}$$

We update the following matrices:

$$\theta^n = \theta^{n-1} + \frac{1}{\gamma^n} B^{n-1} x^n \varepsilon^n, \quad \varepsilon^n = y^n - \theta^{n-1} x^{n-1}$$

$$B^n = B^{n-1} - \frac{1}{\gamma^n} (B^{n-1} x^n (x^n)^T B^{n-1})$$

$$\gamma^n = 1 + (x^n)^T B^{n-1} x^n$$

$$\Sigma^{\theta,n} = \Sigma^{\theta,n-1} - \frac{1}{\sigma_\epsilon^2 \gamma^n} (\Sigma^{\theta,n-1} x^n (x^n)^T \Sigma^{\theta,n-1})$$

Tested Policies:
Knowledge Gradient with Correlated Beliefs
Upper Confidence Bound
Pure Exploitation
Constrained Exploration

Experimental Procedure:
We will test the policies for three distinct time periods of the business day: Morning Rush (7:00 AM - 9:00 AM), Evening Rush (5:00 PM - 8:00 PM), and Graveyard Shift (4:00 AM - 5:00 AM). We will choose the price every 10 minutes and we will simulate each time period over one month (20 business days), giving us a budget of 240 for Morning Rush, 360 for Evening Rush, and 120 for Graveyard Shift.

Each time period has a different true revenue function, which we found earlier in **2.1.1 Offline Lookup Table**. We take these revenue functions and fit a quadratic function around each function's maxima to obtain the true linear approximations for a logistic belief model. Using these approximations, we can utilize our linear belief model for simulations.

For each simulation, we will start with one of the time periods (splitting it up into three problems). We will set the revenue function for that time period. We will then loop through N trials, where N is the budget for that time period. For each iteration, the observation is found by

$$R^n = R(P) + \epsilon^n, \ \epsilon^n \sim N(0, \sigma^2)$$

and we will update the belief model. After our budget has been exhausted, we return the value of the function and the final opportunity cost:

$$\Omega = \left| \max_{p \in P} \theta_p^N - \max_{p \in P} \mu_p \right|$$

We will run the simulation T times (100 for UCB, IE, Exploitation, Exploration; 10 for KG due to processing power constraints) and obtained the average final opportunity cost:

$$\overline{\Omega} = \frac{1}{T} \sum_{t=1}^{T} \Omega_t$$

2.2.2 Online Linear Belief Model

We follow the same principles as offline, except with the online objective function. Our simulations will also be online.

Objective Function

$$\zeta = \max_{p \in P} \mathbb{E}^{\pi} \sum_{n=0}^{N} \gamma^n R^n$$

where $\gamma = 1$ in our case, since each day is worth the same to us.

Alternatives:
Refer to Alternatives of **2.2.1 Offline Linear Belief Model**

Observed Values:
Refer to Observed Values of **2.2.1 Offline Linear Belief Model**

Belief Model:
Refer to Belief Model of **2.2.1 Offline Linear Belief Model**

Prior:
Refer to Prior of **2.2.1 Offline Linear Belief Model**

Updating:
Refer to Updating of **2.2.1 Offline Linear Belief Model**

Tested Policies:
Knowledge Gradient with Correlated Beliefs
Upper Confidence Bound
Pure Exploitation

Constrained Exploration

Experimental Procedure:
Refer to Experimental Procedure of **2.2.1 Offline Linear Belief Model.**

However, instead of comparing opportunity costs, we store a value for the cumulative revenue. Following each sample iteration, we add the observed revenue to the cumulative revenue. We will compare policies using cumulative revenue instead of opportunity cost.

3. Experimental Policies (Algorithms)

For offline learning, we will use Pure Exploitation as a baseline and compare it with Constrained Exploration, Interval Estimation, and Knowledge Gradient with Correlated Beliefs. For our offline Linear Belief Model, we will be using Upper Confidence Bound in place of Interval Estimation.

For online learning, we will use Pure Exploitation again as a baseline and compare it with Constrained Exploration, Upper Confidence Bound, and Online Knowledge Gradient with Correlated Beliefs.

3.1 Pure Exploitation

The pure exploitation heuristic always chooses the price P^{π} that we estimate to yield the highest revenue at that point in time. The policy is described by:

$$p^n = \arg\max_{p \in P} \mu_p^n$$

where μ_p^n represents our belief about the revenue yielded by price *p* at time *n*.

We use pure exploitation as a baseline policy against which we compare the other policies. It is generally more useful in an online setting than offline; in offline, ignoring uncertainty can cause our learning process to stagnate around a suboptimal subset of the truth. As a result of the stagnation, it can be difficult to lower opportunity costs. Exploitation ignores uncertainty and correlated beliefs, which makes it suboptimal for our belief model.

3.2 Constrained Exploration

We base our constrained exploration policy on the pure exploration heuristic. This heuristic assigns an equal probability to each alternative, and the price p^n is chosen at random given this probability distribution.

We see pure exploration as an effective policy in our offline Lookup Table and Linear Belief Models, as it allows for more data points to which we fit our estimated curve. However, in an online situation, pure exploration is relatively ineffective; we prioritize cumulative revenue and pure exploration often results in poor revenue for any given sample period.

We constrain pure exploration in two ways. (1) We do not select $p^n = \$0.00$, because it invariably results in zero revenue—and the government knows this. (2) In our Linear Belief Model, we constrain exploration to only choose from the domain of price alternatives in which the estimated revenue is nonnegative, as negative revenues cannot logically occur during the sampling. This is a defect on the part of our linear belief model; because we assume a base truth that is logistic in nature, our quadratic approximation of the maximum only covers a small subset of possible prices and results in a steep decline into negative revenues. We acknowledge the main drawback of our second constraint—if our prior is narrower than the true quadratic function, then our exploration will ignore some positively-valued prices. However, our belief model is based on the belief that those points outside of the prior will lead to true revenues far below the maximum. Thus, the drawback is only relevant if our prior is significantly narrower than the truth. We assert that the government is confident enough in the relative accuracy of its prior to accept this possible weakness.

We note that given these constraints, the explored alternative is still chosen based on a uniform probability distribution over the valid subset of alternatives.

3.3 Interval Estimation

Interval estimation chooses the sample price as such :

$$p^n = \arg\max_{p \in P} \mu_p^n + z_\alpha \sigma_p^n$$

We have z_α as a tunable parameter and σ_p^n as the standard deviation of the distribution of our belief at the time n. The term $z_\alpha \sigma_p^n$ represents our "uncertainty bonus", which is higher for alternatives which have been infrequently chosen or not chosen at all.

This policy depends heavily on correct tuning of z_α. We tune the policy and then use it as one of the main competitors in the offline Lookup Table model, as interval estimation has been empirically shown to perform very well.

3.4 Modified Upper Confidence Bound

The problem with normal UCB policies is that they experiment with every alternative before balancing between exploration and exploitation. Our budgets are 120, 240, and 360 and we have 201 alternatives. Obviously it would not be efficient to try all 201 prices in an online environment with such a small budget. So we decided to create our own modified UCB policy.

First, we explore over the entire range of prices where we are searching for the maximum revenue ($0 to $20 for AM, $0 to $10 for AM, and $0 to $2 for Graveyard shift). Doing this allows us to make sure our prior is fairly accurate before we move on to the actual UCB policy. But we still have the problem that UCB tries every alternative at least once. We must decrease the range of alternatives – even more than the assumptions above. To do this, we find the variance of the revenues observed from the first step of exploration. We then create a "net" around our current best estimate of a price that maximizes revenue. Higher variance in exploration means a wider net. Hopefully, the "net" would narrow the alternatives enough to use UCB while still "catching" the true maximum somewhere in the range.

Then, we run UCB in this narrow range. We found that an exploration of 30 trials at the beginning to be the right amount. Our net size was the prices that correspond to 1/4 of the standard deviation of revenues above and below our estimate maximum. So our UCB runs in 3 distinct steps. First, exploration. Then, exploration (try every alternative) within our net. Then, the UCB mostly exploits.

This policy is only really an advantage when we have a decent prior. If we had an uninformative prior, we could "catch" that during the first exploration step seeing how much our estimates changed. In this case, the net would be too big and we would be back to a standard UCB policy that tries everything. If our prior were wrong and we didn't explore enough to begin with, we would cut off the true maximum from our future search. However, we found a budget of 30 for this stage to be enough for a relatively accurate prior.

3.5 Knowledge Gradient with Correlated Beliefs

As our belief model relies on correlated beliefs, we use the Knowledge Gradient policy with Correlated Beliefs. The offline KGCB value is computed as such (5.3 in *Optimal Learning*):

$$\nu_p^{KGCB,n} = \mathbb{E}_p^n[\max_{p' \in P} \mu_{p'}^{n+1} - \max_{p' \in P} \mu_{p'}^n]$$

For the sample period *n*, we define two vectors:

$$a_p^n = \mu^n$$

$$b_p^n = \frac{\Sigma^n e_{p^n}}{\sqrt{\lambda^W + \Sigma_{pp}^n}}$$

We use these in the function:

$$h(z) = a_p^n + b_p^n z$$

which generates a family of lines. We order this set of lines by slope (b_p^n) and remove dominated lines. The intersection points of successive pairs of remaining lines will be termed c_p^n. The KGCB index is then found:

$$\nu_p^{KGCB,n} = h(a,b) = \sum_{i=1}^{M} (b_{i+1} - b_i) f(-|c_i|)$$

$$f(z) = z\Phi(z) + \phi(z)$$

For our online simulations, we choose thusly ($\gamma = 1$):

$$p^n = \arg\max_{p \in P} \mu_p^n + (N - n)\nu_p^{KGCB,n}$$

4. Experimental Design

4.1 Simplifying Assumptions

1. *Two-Laned Freeway*: We group all of the regular lanes on the freeway into one "regular" set, and we group all of the pay lanes into one "pay lane" set. This way, we can model general demand for pay lanes and we do not consider separate demands for each lane. Each lane is, for all intents and purposes, physically identical

2. *Constant Overall Traffic:* We assume a constant number of cars traveling through the entire freeway during a certain time period. The number of cars that decide to pay for the pay lane is variable, but there is a hard cap as established by our traffic data.

3. *Limited Alternative Range*: We consider possible alternatives to be \$0.00 or greater and less than or equal to \$20.00.

4. *Constant Driving Conditions*: We assume there are no weather or external factors that significantly impact the traffic (i.e., hurricane, quarantine, sporting event)

5. *Business Days*: We assume all of our measurement to take place during business days.

6. *No Maintenance Costs:* We assume that the government pays no cost for maintaining the freeway, and thus that revenue observed must be non-negative.

4.2 Experimental Setup

4.2.1 Prior Construction

For the **Lookup Table Belief Model**, our prior is generated by the following function:

$$\theta_p^0 = \begin{cases} 0.10p\left(-\dfrac{\beta}{10P_{max}}p + \beta\right) & 0 \le p \le 10P_{max} \\ 0 & 10P_{max} < p \le 200 \end{cases}$$

This is constructed from two key pieces of information:

1. Demand at x=0 (P=\$0.00) is equal to half of the overall traffic. At P=\$0.00, the pay lanes and regular lanes are effectively indistinguishable. Therefore, the drivers have no preference regarding the pay lanes and half of all traffic takes the pay lane.
2. The demand decreases at price increases, reaching 0 at some maximum price. The government assumes that the demand decreases linearly with price.

There is an implicit assumption that revenue is equal to \$0 at P=\$0.00, as regardless of the number of cars taking the pay lane, nobody pays anything. The government knows the value of β, as it has access to its own traffic data.

For AM Rush Hour, the government assumes \$20.00 for the maximum price; for PM Rush Hour, the government assumes \$10.00 for the maximum price; for Graveyard Shift, the government assumes \$2.00 for the maximum price.

Our prior covariance matrix is generated by the covariance function:

$$Cov^0\big(R(P), R(P')\big) = \sigma^2 e^{-\alpha|P-P'|}$$

where $\sigma^2 = Var^0(R(P))$. σ^2, α are adjusted to fit public traffic data from the California Department of Transportation.

For the **Linear Belief Model**, our prior is generated using a different method. The government first creates a family of quadratic functions that it believes may reasonably approximate the area near the maximum of the true revenue function:

$$\begin{aligned} R_1(P) &= \theta_1^1 + \theta_2^1 P + \theta_3^1 P^2 \\ R_2(P) &= \theta_1^1 + \theta_2^1 P + \theta_3^1 P^2 \\ R_3(P) &= \theta_1^1 + \theta_2^1 P + \theta_3^1 P^2 \\ R_4(P) &= \theta_1^1 + \theta_2^1 P + \theta_3^1 P^2 \end{aligned}$$

We then create our prior estimates of each parameter:

$$\theta_k^0 = \frac{1}{4}\sum_{n-1}^{4} \theta_k^n$$

Our covariance matrix for the parameters also comes from the family of functions:

$$Cov^0\big(\mu_i, \mu_j\big) = \frac{1}{N-1}\sum_{k=1}^{N} (x_{ki} - \bar{x}_i)(x_{kj} - \bar{x}_j)$$

The priors and families of functions are listed below in **4.2.3 Data and Parameters**.

4.2.2 Truth Generation

For the **Lookup Table Belief Model**, we take our true demand function to be logistic, and our true revenue function is:

$$R(P) = \frac{P \cdot M}{1 + e^{-\mu_1 + \mu_2 P}}$$

$$M = \frac{\beta}{2}(1 + e^{-\mu_1})$$

β being the peak traffic over the expressway during the sample period. When we generate an observation, we take the value of the true revenue function at that price and add measurement noise:

$$R^n = R(P) + \epsilon^n,\ \epsilon^n \sim N(0, \sigma^2)$$

For the **Linear Belief Model**, we initially wanted to use the same true demand function and use the same observation method. However, we realized that observations near the fringes would significantly affect our quadratic function, since the logistic-based true revenue is only approximately quadratic at or near the maximum. As a result, we are using a quadratic approximation of the true revenue curve as the truth for the linear belief model.

We create this curve by selecting a set of 6 points at or near the maximum of the logistic-based true revenue curve and running a quadratic regression to create:

$$R(P) = \mu_1 + \mu_2 P + \mu_3 P^2$$

We then generate a set of values of the revenue function for each alternative:

$$\begin{bmatrix} 1 & P_1 & P_1^2 \\ \vdots & \vdots & \vdots \\ 1 & P_{201} & P_{201}^2 \end{bmatrix} \begin{bmatrix} \mu_1 \\ \mu_2 \\ \mu_3 \end{bmatrix} = \begin{bmatrix} R(P_1) \\ \vdots \\ R(P_{201}) \end{bmatrix}$$

and add measurement noise:

$$R_k^n = R(P_k) + \epsilon^n,\ \epsilon^n \sim N(0, \sigma^2)$$

Note that since we are using a quadratic truth approximation, this can return negative revenue. Realistically speaking, we cannot observe negative revenue, as the pay lanes are already built, and revenue collected is always nonnegative. However, we run into a problem here—we tried setting the observed value to 0 if the returned observation was negative, but this tends to deform the estimated curve. We will explore the problem in more detail in **6.2 Major Limitations**.

4.2.3 Prior Data and Parameters

Simulation Parameters

AM Rush Hour Budget: 240
PM Rush Hour Budget: 360

Graveyard Shift Budget: 120
Total # of Alternatives: $K = 201$
Upper Confidence Bound Exploration Budget: 30

AM Rush Hour (7-9 AM)

Overall Sample Traffic (β) = $\frac{21500}{6} \approx 3583$ Cars

Lookup Table Model

Logistic Parameter $\mu_1 = 4.6$
Logistic Parameter $\mu_2 = 0.46$
Revenue Standard Deviation $\sigma_M = 1500$
True Maximum Revenue $R_{max} = \$10355.22$
True Optimal Price $P^* = \$7.90$
Covariance Function Parameter $\alpha = 0.30$

Linear Belief Model

Linear Parameter $\mu_1 = -5988.57$
Linear Parameter $\mu_2 = 4164.45$
Linear Parameter $\mu_3 = -265.83$

Linear Belief Prior Generation

$R_1(P) = -5000 + 4000P - 280P^2$
$R_2(P) = 0 + 6000P - 800P^2$
$R_3(P) = 5000 + 1500P - 100P^2$
$R_4(P) = -45000 + 15000P - 1000P^2$
Point Estimate $\theta_1^0 = -11250$
Point Estimate $\theta_2^0 = 6625$
Point Estimate $\theta_3^0 = -545$

Interval Estimation Tuned $z_\alpha = 3.0$

PM Rush Hour (5-8 PM)

Overall Sample Traffic (β) = $\frac{21500}{9} \approx 2389$ Cars

Lookup Table Model

Logistic Parameter $\mu_1 = 3.7$
Logistic Parameter $\mu_2 = 0.82$
Revenue Standard Deviation $\sigma_M = 800$
True Maximum Revenue $R_{max} = \$2991.70$
True Optimal Price $P^* = \$3.60$
Covariance Function Parameter $\alpha = 0.30$

Linear Belief Model

Linear Parameter $\mu_1 = -995.07$
Linear Parameter $\mu_2 = 2171.69$

Linear Parameter $\mu_3 = -296.47$

Linear Belief Prior Generation

$R_1(P) = -500 + 2000P - 300P^2$
$R_2(P) = 0 + 3500P - 850P^2$
$R_3(P) = 1200 + 850P - 100P^2$
$R_4(P) = -13600 + 10000P - 1500P^2$
Point Estimate $\theta_1^0 = -3225$
Point Estimate $\theta_2^0 = 4087.5$
Point Estimate $\theta_3^0 = -687.5$

Interval Estimation Tuned $z_\alpha = 3.0$

Graveyard Shift (2-4 AM)

Overall Sample Traffic (β) = $\frac{2150}{6} \approx 358$ Cars

Lookup Table Model

Logistic Parameter $\mu_1 = 3.1$
Logistic Parameter $\mu_2 = 2.82$
Revenue Standard Deviation $\sigma_M = 40$
True Maximum Revenue $R_{max} = \$107.46$
True Optimal Price $P^* = \$0.90$
Covariance Function Parameter $\alpha = 0.30$

Linear Belief Model

Linear Parameter $\mu_1 = -33.19$
Linear Parameter $\mu_2 = 302.82$
Linear Parameter $\mu_3 = -162.99$

Linear Belief Prior Generation

$R_1(P) = -50 + 300P - 150P^2$
$R_2(P) = 0 + 350P - 300P^2$
$R_3(P) = 75 + 75P - 50P^2$
$R_4(P) = -400 + 1000P - 500P^2$
Point Estimate $\theta_1^0 = -93.75$
Point Estimate $\theta_2^0 = 431.25$
Point Estimate $\theta_3^0 = -250$

Interval Estimation Tuned $z_\alpha = 2.5$

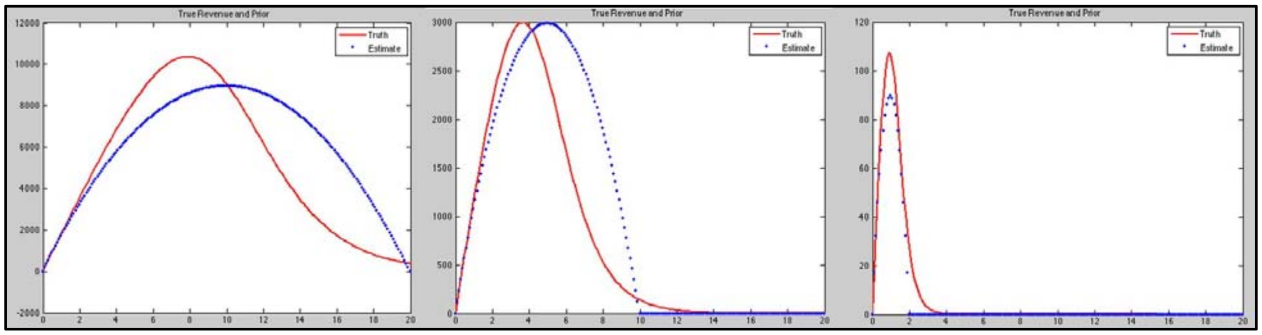
5. Experimental Results

In our experimental results section, we will compare the different policies. Instead of just presenting the results and all of the graphs of every experiment we will try to write the results section like a story, including the pertinent and interesting graphs as they come about. We have already explained our belief model and why we chose to model the problem in these ways. In the results section, we will explore (pun point) what worked and what didn't. We will interpret our results in both offline and online, and in all three scenarios: AM, PM, and Graveyard. There will be differences in all of these scenarios and we will highlight these differences and explain why they occurred. The results section is going to include not only our results, but also our interpretations, our limitations, and our recommendations. It is in this way that the results section will be the bulk of our project – with our conclusion only summarizing the most important parts of the results and wrapping up the project with a "bigger picture" outlook.

5.1 Lookup Table

First, we will look at our lookup table implementation. You may recall that we created our truths from public traffic data online. We created our priors from the two assumptions that (1) the government knows only that demand of the pay lane is half of total traffic at P=0 and (2) the government knows the price (Pmax) at which no cars will be willing to pay. It then creates a linear demand prior between these two points and coverts this linear demand into the revenue prior by just multiplying by price.

The truths and priors for AM Rush Hour, PM Rush Hour, and Graveyard Shifts are as such:



AM Rush Hour PM Rush Hour Graveyard Shift

As the range of advisable prices condenses as we move from the heaviest traffic in AM to the lightest traffic in Graveyard, we see that our prior estimate maximum is closer to the true maximum. This makes intuitive sense as we are working on a smaller scale as we move down. We can expect that our Graveyard shift will find the maximum quicker. We can also expect for Graveyard to be more accurate when we exploit or during online policies and we will explore these thoughts further on.

For each simulation, we created 6 graphs. First, we updated the graphs above and compared our *final* estimate to the truth. Second, we graphed opportunity cost (how far away the max of our estimate is from the true maximum) by iteration. Third and fourth, we made a bar graph of the prices we tested and also a line graph to see if the prices we tested converged to a specific price. Lastly, for the online simulations, we graphed revenue by iteration and cumulative revenue.

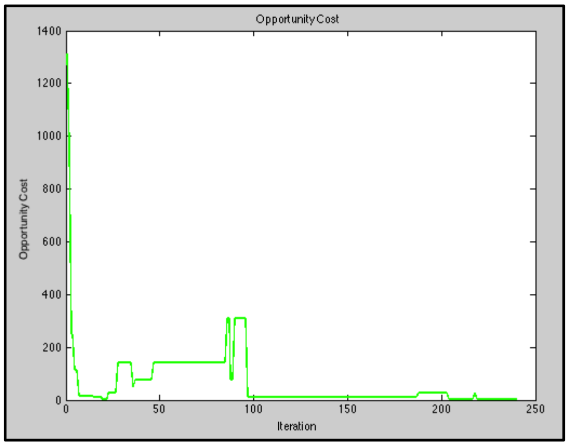
Obviously, some of these graphs will have little to no meaning. Exploration will just have close to a uniform bar graph of prices chosen and will have a meaningless graph of revenue by iteration. We will omit these graphs and only explain the ones with meaning.

First, we will look at offline policies. We will present the average opportunity cost results in a chart at the end. But for now, we will do a qualitative look into one experiment for each situation and policy.

5.1.1 Offline Lookup Table

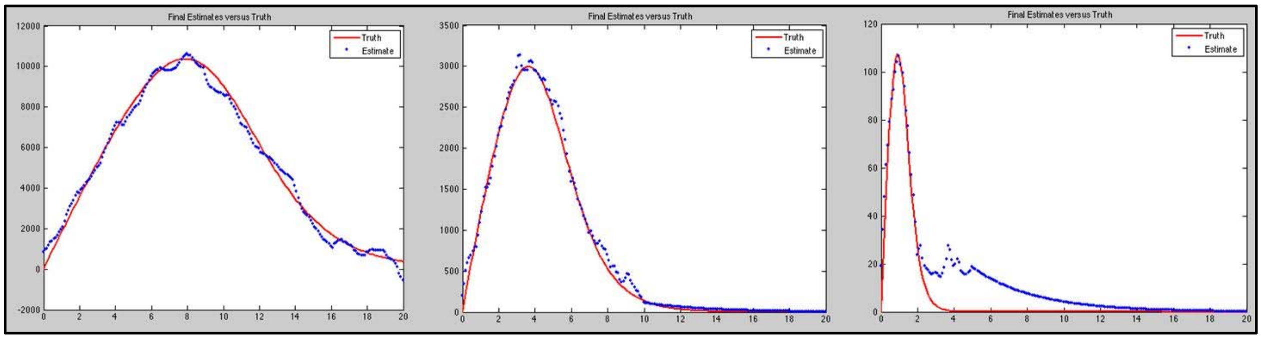
We will now look at how exploration works for each of the three situations. Remember, we allotted a budget of 240 for AM, 360 for PM, and 120 for Graveyard.

Exploration opportunity cost for AM Rush Hour, **Exploration:**



The opportunity cost graphs for PM Rush Hour and Graveyard Shift are similar.

Final estimates compared with the truth for **Exploration**:

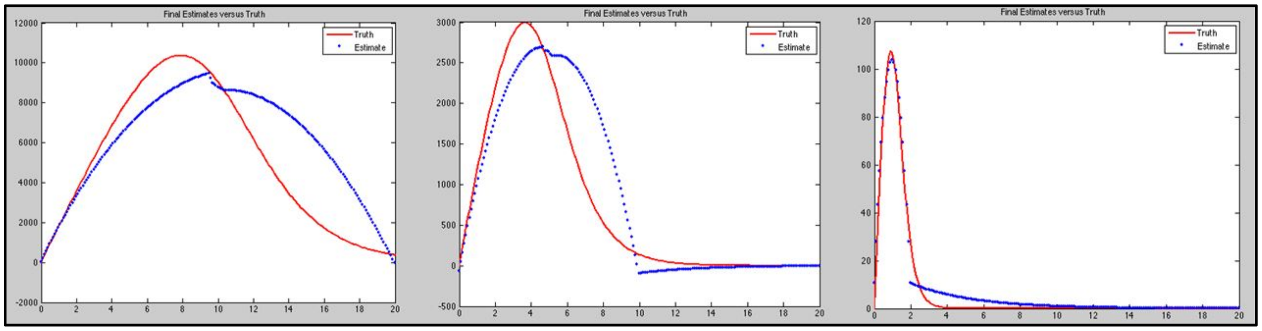


AM Rush Hour PM Rush Hour Graveyard Shift

As you can see, exploration finds the maximum pretty well. This makes sense because our budget is fairly high. Opportunity cost is improved fairly quickly but may not reach zero exactly. Graveyard Shift had the best prior and consequently found the max most often using exploration. The weird squiggly things you see on the right-hand side of PM and Graveyard result from our using correlated beliefs and only exploring in the constrained ranges. Since we assume the government knows that these aren't good prices to charge, the irregularities are not important.

Exploitation is highly dependent on our priors. It is in this way we can see if our estimates can be corrected if we continuously exploit.

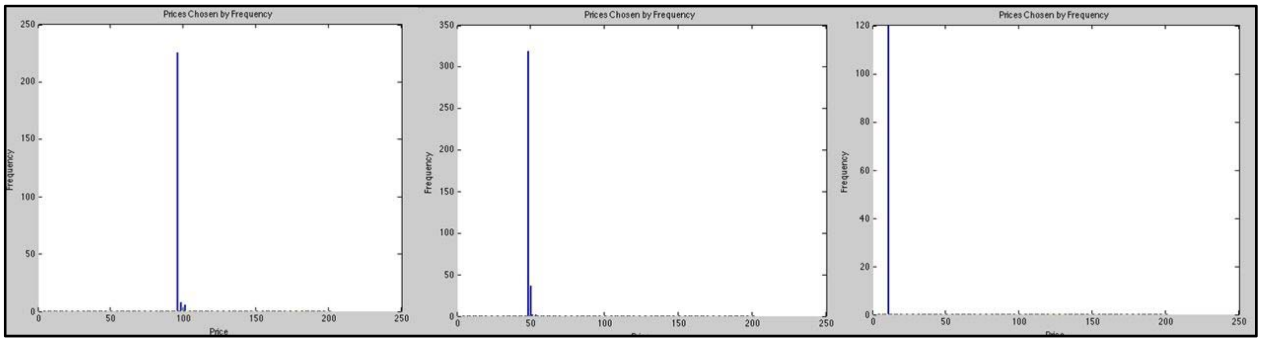
Final estimates compared with the truths for **Exploitation:**



AM Rush Hour PM Rush Hour Graveyard Shift

Only Graveyard shift (with the most accurate prior) turned out well. AM and PM rush hour would test a lower price and observe a higher value. It would then continue to exploit that higher value and never have a higher estimate at any other point. Opportunity costs are high and we do not try many different prices for each.

Prices chosen by frequency, **Exploitation:**

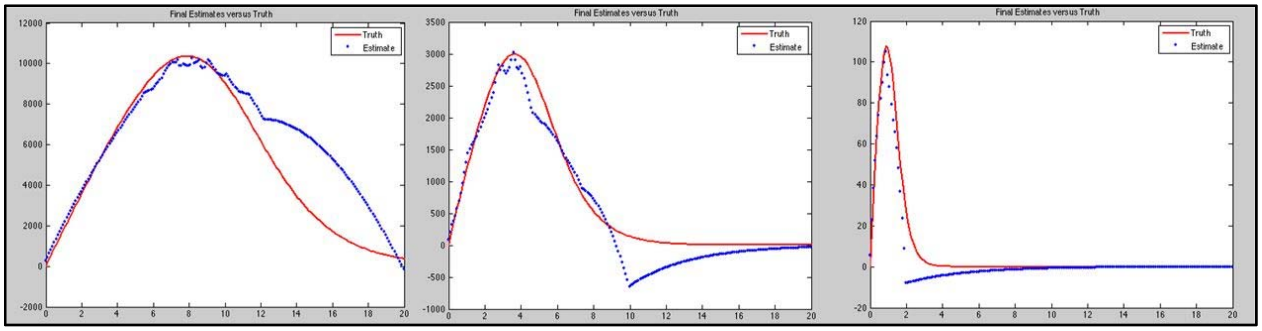


AM Rush Hour PM Rush Hour Graveyard Shift

If we were to imagine what LA actually does when pricing the freeways, we would suspect something like pure exploitation. It chooses what it thinks to be the best and observes a great result and decides to repeat it.

Okay let's move onto a couple more complicated policies.

Final estimates compared with the truths for **Interval Estimation:**

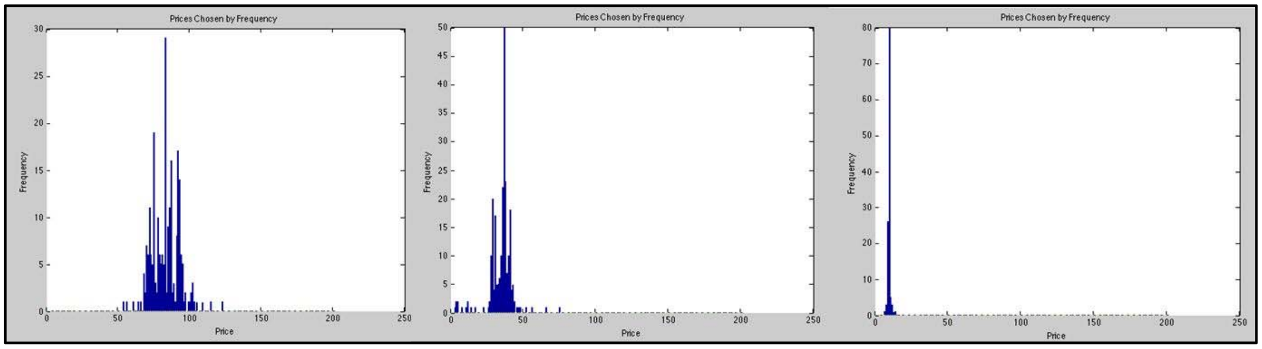


AM Rush Hour PM Rush Hour Graveyard Shift

Interval estimation looks a bit funky. Basically, interval estimation is exploiting but with an uncertainty bonus. In all scenarios, IE worked pretty well. The uncertainty bonus allowed us to try out prices closer to the true max and we eventually figure it out. Opportunity costs were low. The weird tails on the right side of each graph are just products of correlated beliefs. The uncertainty bonus was never enough to try prices that high so those estimates changed up or down depending on the few observations at higher prices. Again, we are only trying to find the maximum, not fit the curve so these are fine.

It is also interesting to see the range of prices we end up trying below. As long as prior was close enough to the truth that the uncertainty "net" included the truth, we ended up finding the truth.

Prices chosen by frequency, **Interval Estimation:**

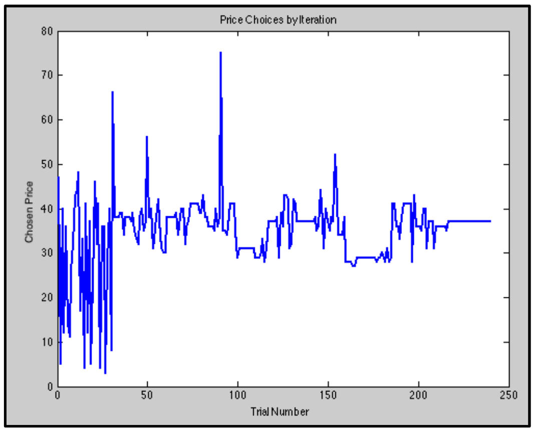


AM Rush Hour PM Rush Hour Graveyard Shift

IE never tried truly bad choices and used the budget wisely to find the truth.

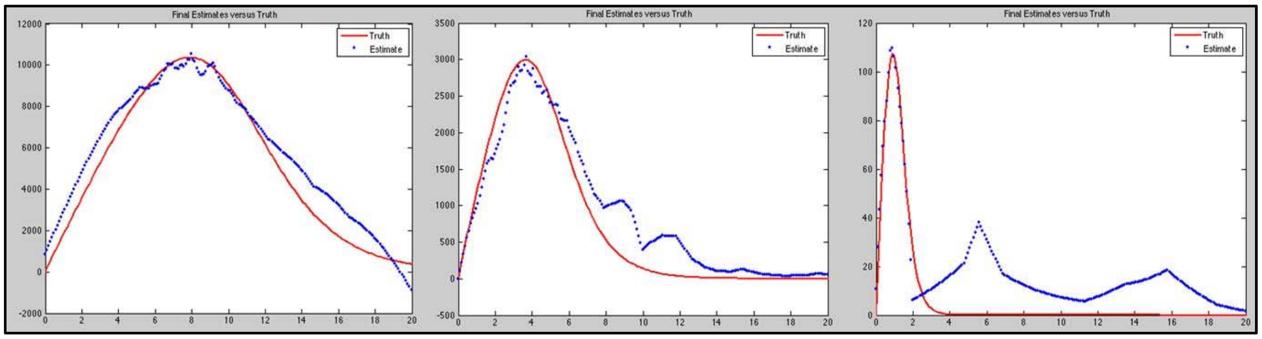
It is also worth mentioning that IE tended to converge in testing prices close to the maximum. The graph of PM below is indicative of the three cases. This is in contrast to offline knowledge gradient as we will see next.

Prices chosen by iteration, **Interval Estimation**, PM Rush Hour:



Offline **Knowledge Gradient**, as you can see below, did not really outperform interval estimation. They were both great, to be sure. Both of the two policies take uncertainty into account and find the maximum (or close to it) with ease. They had similar opportunity costs.

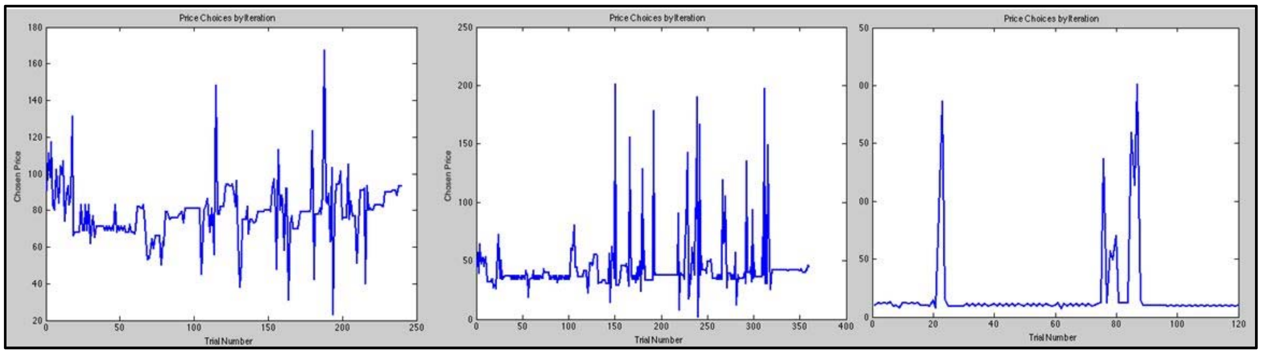
Final estimates compared with the truths for **KGCB, Offline:**



AM Rush Hour PM Rush Hour Graveyard Shift

As opposed to IE, which generally narrowed the range of prices it would search, KG narrows its search initially then explores more later on.

Prices chosen by iteration, **KGCB, Offline:**



AM Rush Hour PM Rush Hour Graveyard Shift

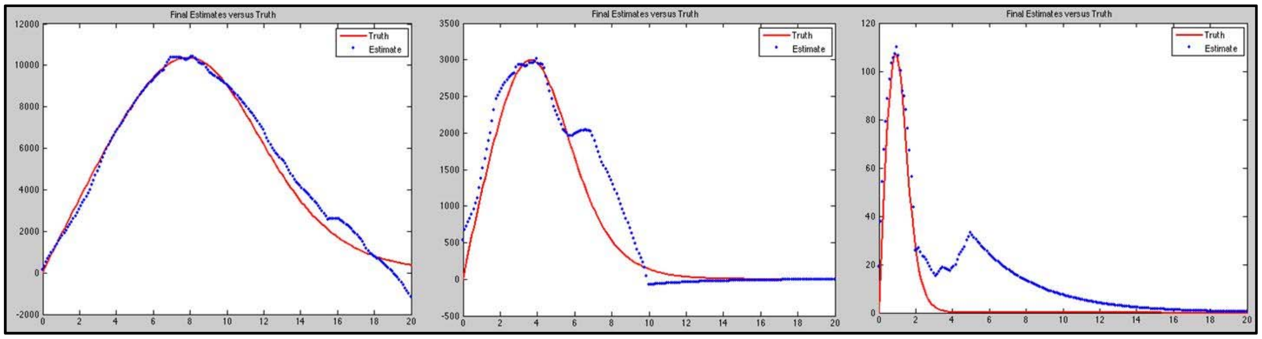
In all three cases, offline KG finds the maximum within the first 50 iterations. Once it is confident in the local maximum, it starts exploring since there is nothing left to learn in the vicinity of the maximum. It got a similar opportunity cost as IE, but got to it differently.

5.1.2 Online Lookup Table

We will now do the same problems, but online. We will also switch offline KG to online KG and switch IE to our own UCB policy. We will include revenue graphs where they are interesting and not include the exploration and exploitation graphs that we did offline.

Exploration and **Exploitation** are pretty intuitive. Since our constrained exploration still surveys most of the area of the function, the exploration average revenue is similar to the average value of the curve. For exploitation, our estimates do not really change so our revenue is the value of the truth at our prior maximum.

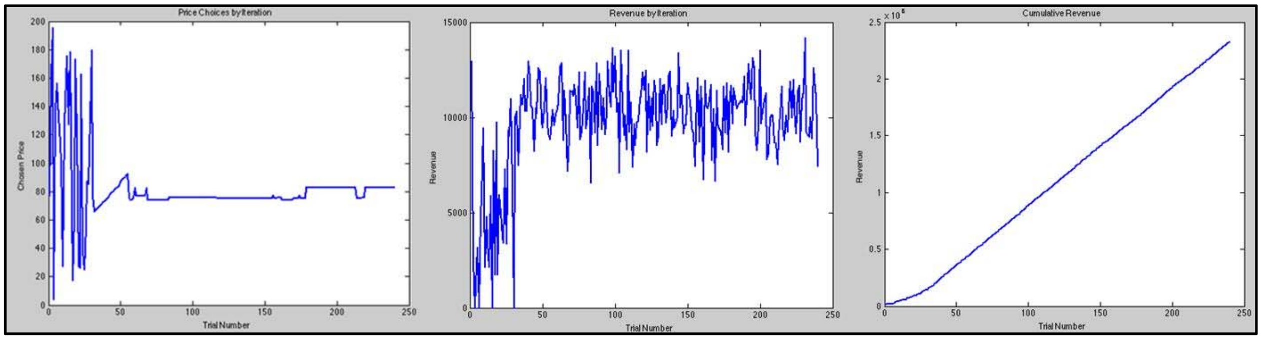
Final estimates compared with the truths for **UCB:**



AM Rush Hour PM Rush Hour Graveyard Shift

As explained earlier, our **UCB** policy works in three broad steps: exploration, exploration near the maximum, then exploitation. You can see the three distinct steps in the AM prices chosen by iteration. In essence, we are manually starting with exploration then converging towards exploitation just like any online policy should. Since the pure exploration in the first 30 of the budget gets bad revenue values, we can see how the revenue increases as we continue into the second and third steps below.

Choice, revenue, and cumulative revenue per iteration for **UCB,** AM Rush Hour:

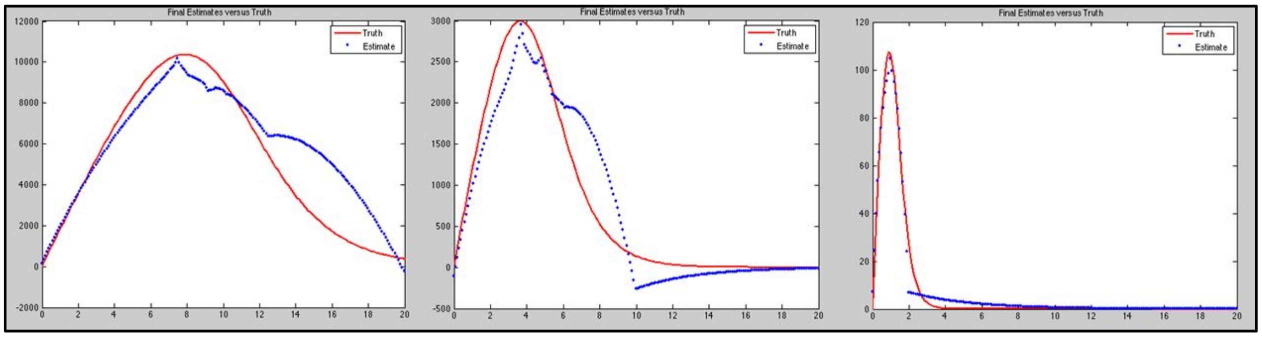


Choice by Iteration Revenue by Iteration Total Rev. by Iteration

It also finds the maximum very well, almost as well as online KG. In the end, because it spends the first 30 iterations of pure exploration, it doesn't perform as well in the online environment as KG.

The first 30 revenues are pretty bad and they are a significant part of whichever budget (120,240,360). Even though the rest of the revenues are very good, UCB only performs about as well as pure exploitation, which is what we're guessing the government was already doing before we decided to take on this project.

Final estimates compared with the truths for **KGCB, Online:**

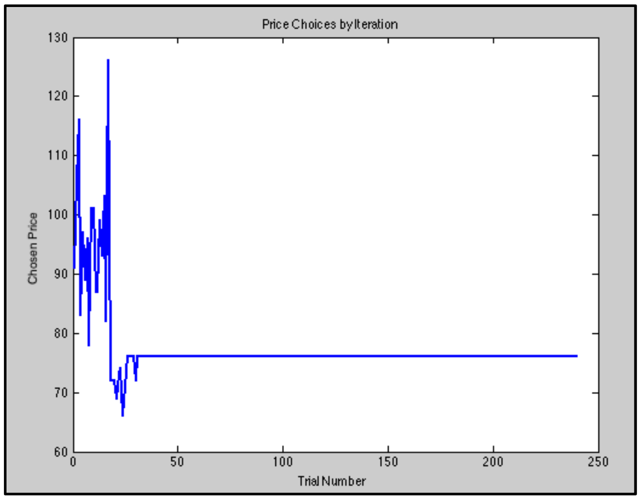


AM Rush Hour PM Rush Hour Graveyard Shift

Again, knowledge gradient performs well in the online environment. The average opportunity cost is definitely higher than offline – there is less exploration and more exploitation. However, out of all of our policies, online KG maximized revenue best for the lookup table model.

Online KG explores but quickly decides on what it believes to be the maximum and then exploits from then on.

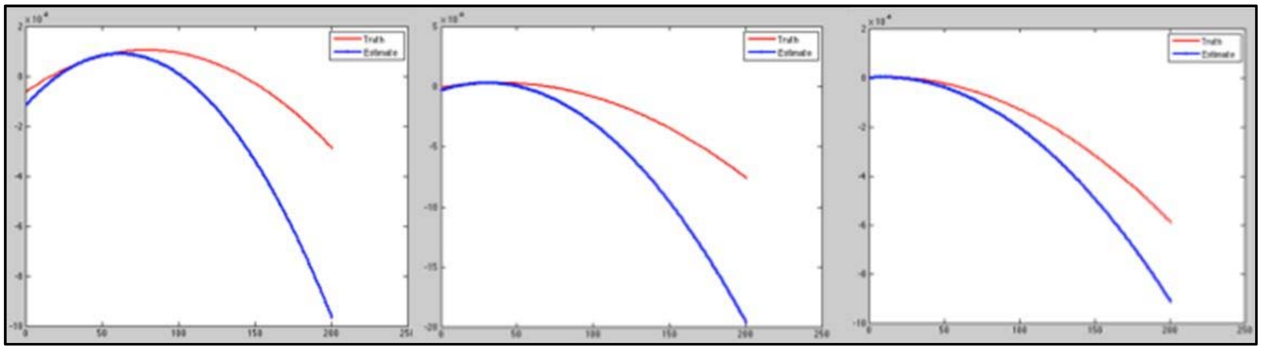
Prices chosen by iteration, **KGCB Online**, AM Rush Hour:



5.2 Local Linear Approximation of Logistic Belief Model

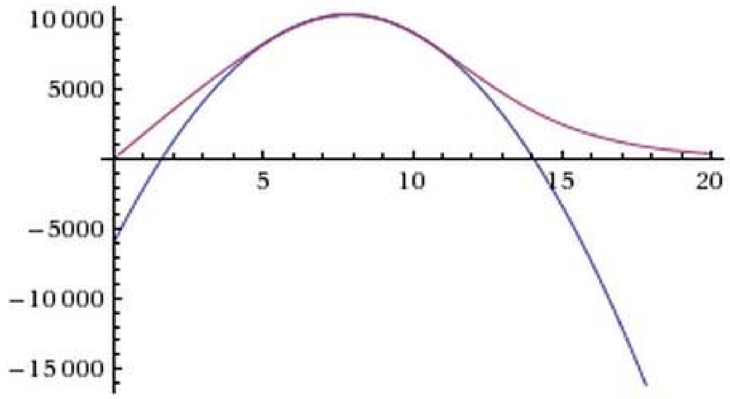
In the linear implementation, we are trying to approximate only the maximum of the logistic truth by using a quadratic function in terms of P. We created the truth by interpreting each of our three logistic truths in the previous section and just finding a quadratic function (linear in theta) that would best fit it near the maximum. We then created our priors using families of curves.

The truths and priors for AM Rush Hour, PM Rush Hour, and Graveyard Shifts are as such:



AM Rush Hour PM Rush Hour Graveyard Shift

So if you look on the y-axis obviously this is showing mostly negative values of revenue, which have no meaning (and are impossible) for us. Negative values for revenue were a huge problem for us in this section and somewhat impeded our progress along the way. Basically, in order to use a quadratic function to fit the top of a logistic curve over a wide-enough interval, it naturally means the quadratic function will fall much quicker than the logistic curve and become negative in our range of prices. See the picture below of our translation of the AM truth:



So, it was a huge problem for us. Basically there were 3 choices. None of them were perfect. First, we could limit the number alternatives into the range where the truth was positive. This was insufficient – we would assume the government knows too much. In this case it would only have prices $2-$14 available to it. But we would be fitting our model based on a truth that we are making up and basically, this wouldn't have been good. The government does have the choices \$0-\$2 and \$14-\$20 available to it. We can't just ignore that.

The second choice was anytime we observe negative revenue; just change the negative revenue to zero. This makes intuitive sense. The minimum revenue we can observe is zero when no one uses the lane. But when we observe \$0 at say P=\$18 in the above graph. It drastically shifts the curve up and basically messes everything up.

It is worth mentioning that this is worse for (1) offline and (2) exploratory policies. This is because the observations are artificially altered most when they are most negative. Because this is a quadratic function, this is to the far left and the far right. For pure exploitation, this is never a problem. For online KG, it is rarely a problem too.

Lastly, we could just cut our losses and allow ourselves to observe negative revenue. We will adjust our policies so that we choose negative revenues as little as possible but will not disallow them in the way that limiting the amount of alternatives would. We are not going to justify this decision in real life reasoning. We tried both setting observations to zero and allowing them to be negative and we switched throughout. Basically, it is not a huge problem for online. For offline, we would quickly decide that the negative observations cannot be the maximum so we wouldn't observe them. The biggest problem is that since this is a quadratic function, KG likes to choose where it would learn the most – precisely in the places where the truth is most different from the prior. Since it is all connected (unlike lookup table) an observation at P=\$18 would drastically change the entire function, and this was the problem. So you will see that we added numerous workarounds, but we admit imperfections in this model.

5.2.1 Offline Linear Belief

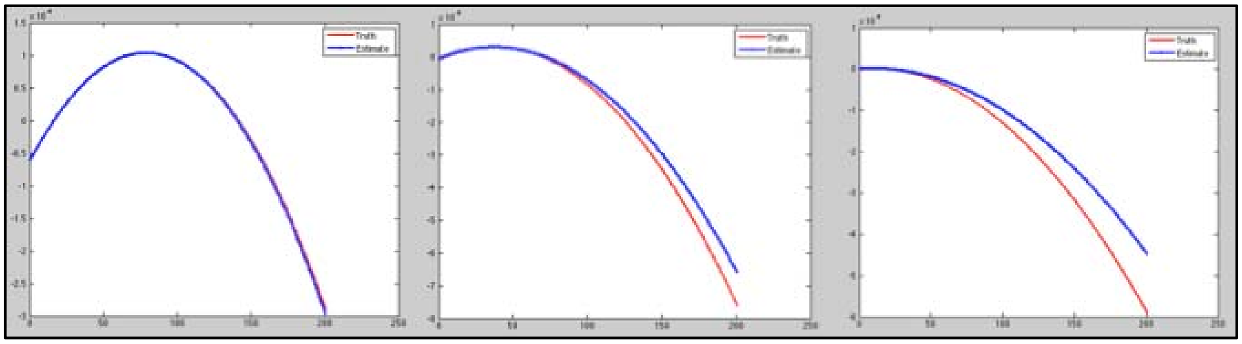
So, for our first work-around, we limited our **exploration** even further. We limited exploration to only values of P where we expected positive revenue. We note that our estimates are not the same as the truth and we are still allowing ourselves to choose some prices P that have negative revenue.

But we are assuming that are estimates are close enough to the truth to minimize the chances of choosing a negative revenue. We found this to be a both realistic and clever workaround. We didn't adjust it further.

But if we had an uncertain or inaccurate prior, then we could limit the exploration even further inside the boundary P's to even further minimize the chance of observing negative revenue.

With our constrained exploration, we had fantastic results. This was not a given, nor was it only a product of our workarounds. It was basically a testament to our linear model. Our opportunity cost was basically 0. It found the function around the maximum perfectly.

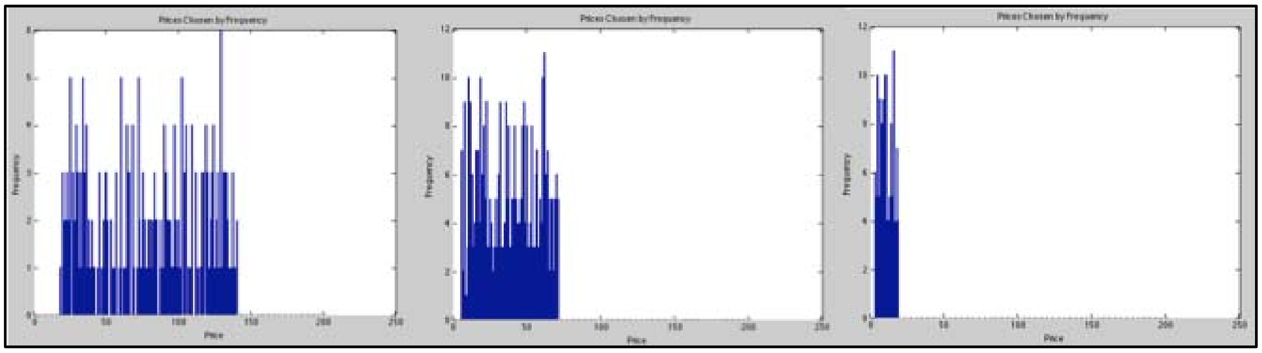
Final estimates compared with the truths for **Constrained Exploration:**



AM Rush Hour PM Rush Hour Graveyard Shift

And we can see how the exploration was constrained in the below graphs.

Prices chosen by frequency, **Constrained Exploration:**

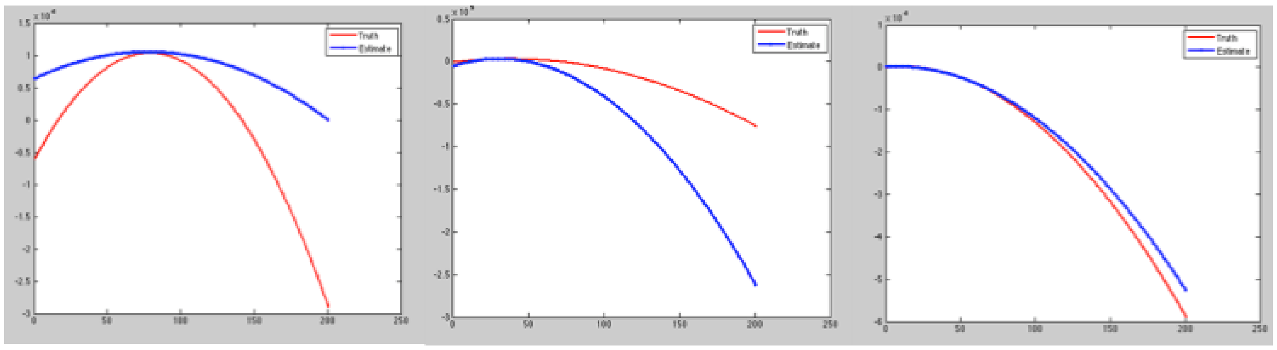


AM Rush Hour PM Rush Hour Graveyard Shift

The reason the function does not fit perfectly for PM and Graveyard is simply that we constrain ourselves to not explore over there so it won't fit (but we won't observe negative revenues often either).

The above problem did not affect **exploitation** since exploitation only chooses positive revenues.

Final estimates compared with the truths for **Exploitation:**

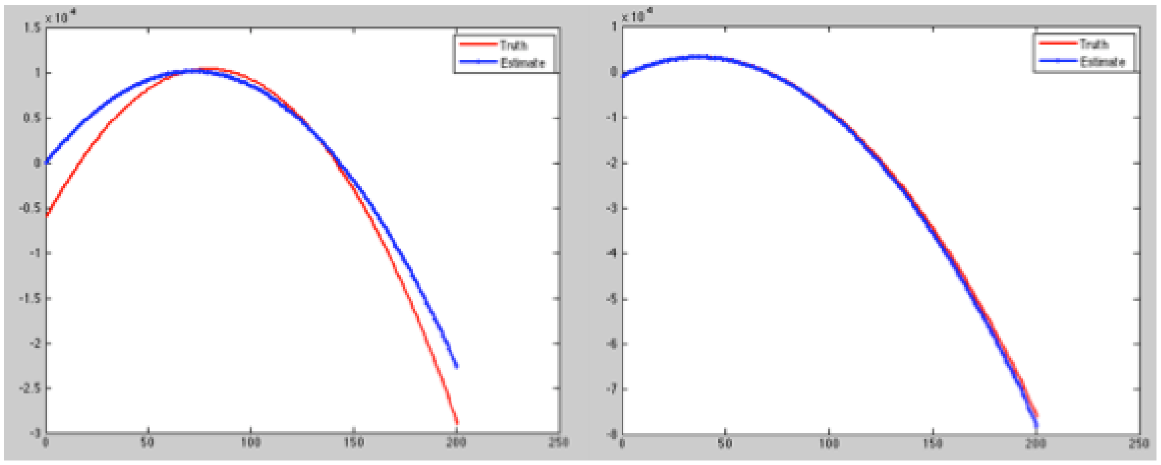


AM Rush Hour PM Rush Hour Graveyard Shift

Exploitation measured the same cluster of points for each of the three situations. Because of this, correlated beliefs raised the estimates of all other prices, which is why you see raised blue lines. Exploitation was hit and miss again, basically dependent on the prior. Also, testing a cluster of points is not the most efficient way of fitting a curve as testing the endpoints would be more helpful. But it turned out decently anyway.

We replaced IE with UCB and we will cover UCB once we get to online, but for now, let us consider offline **Knowledge Gradient**. Offline KGCB ended up being poor for AM but better for PM and graveyard.

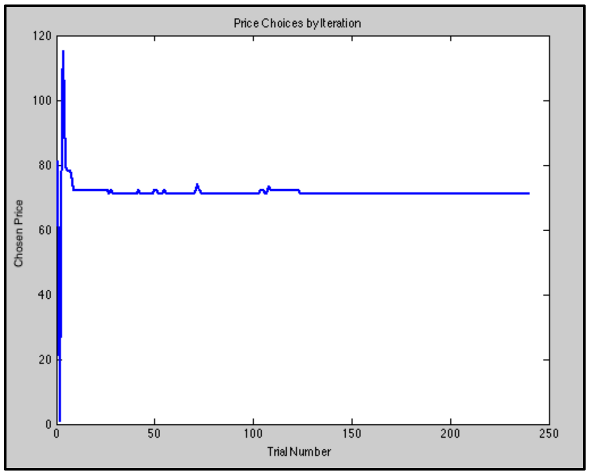
Final estimates compared with the truths for **KGCB, Offline:**



AM Rush Hour PM Rush Hour

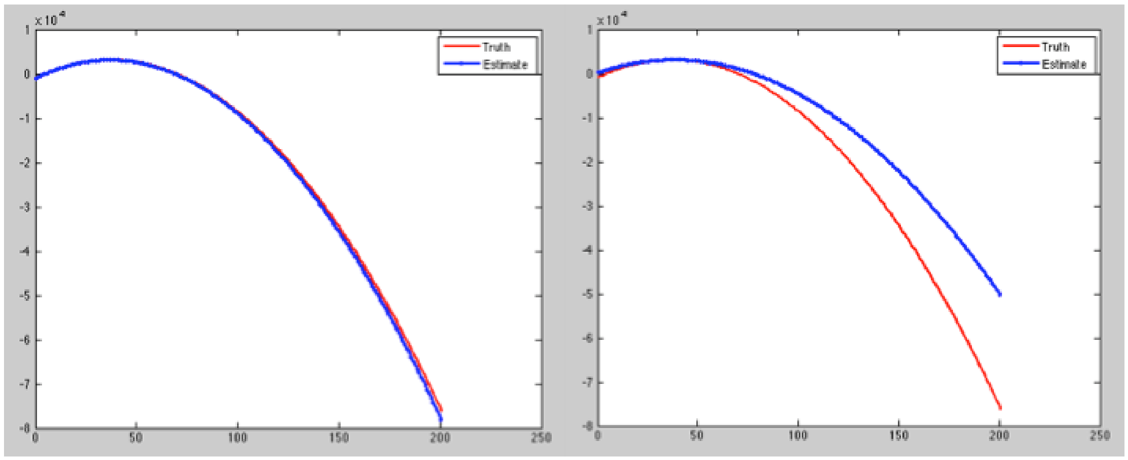
We note that the graph for Graveyard Shift looked nearly identical to PM Rush Hour.

Prices chosen by iteration, **KGCB, Offline:**



Here just for conversation's sake, I will present the difference (for PM) between allowing us to observe negative revenues and setting all negative revenues to zero.

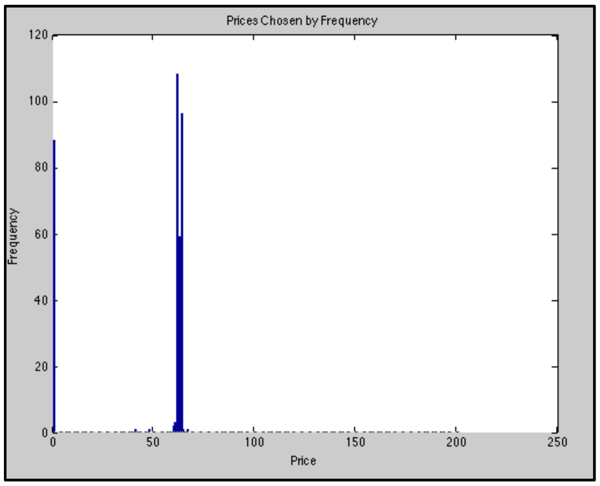
Final estimates compared with the truth for **KGCB, Offline** for PM Rush Hour:



Allowing negative revenue Observations ≥ 0

The fit is much better for negative revenues. Since low prices correspond to zero revenue or higher. The y intercept on the graph on the right is greater than zero. There are also small other differences. The problem is that KG tested the below choices to learn most about the function. This would be okay if we were allowed negative revenues, but in an online environment, this would give us huge negatives or many zeros depending which rule you use.

Prices chosen by frequency, **KGCB, Offline** for PM Rush Hour:



The reason this wasn't terrible, like I alluded to at the beginning of the section, is that this is the result of another one of our workarounds. Again, we constrained KG to choose only among current positive estimates. In this case P=0 was a positive estimate (incorrect in real life) so we were allowed to observe it. With our work-around, it would have been possible to set negative revenues to zero, but new complications would have arisen during online.

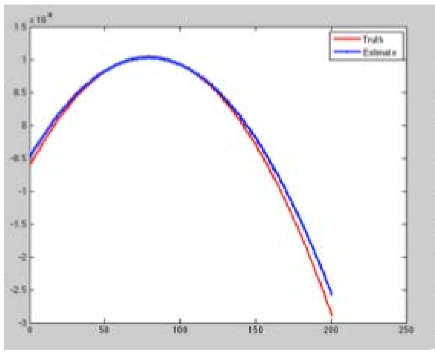
5.2.2 Online Linear Belief

Constrained Exploration was its normal self. Since we constrained it (not overly so though) to only pick decent prices, it did a little better than exploration in the lookup model by comparison. Otherwise, it does terribly.

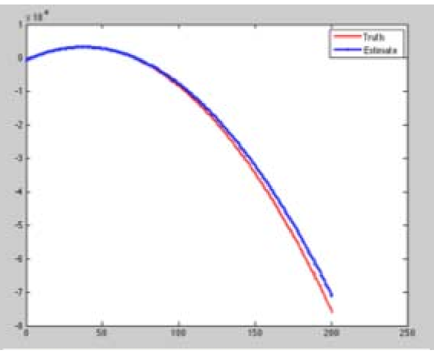
Exploitation does surprisingly well. It competes with KG for AM and PM and beats it in the Graveyard Shift. This is because it doesn't have any negative revenue observations in the way the KG would explore at the beginning in online. Exploitation, in short, is a great choice when your model is only accurate near the true maximum – like our quadratic approximation of the logistic curve.

The **UCB** had some weird results. It did great for AM. Even after losing out on a lot of revenue in the exploration phase, it almost caught up to KG and the average opportunity cost was much lower than online KG. This means that if we had a larger budget, UCB would have won. For PM, it seems as though the initial budget of 30 for exploration might not be sufficient. For Graveyard, the budget of 30 is too much of the 120 total in the budget to be efficient. If UCB were tuned perfectly, and I'm not convinced it could be without knowledge of how accurate your prior is, then it would easily be the best linear, online policy.

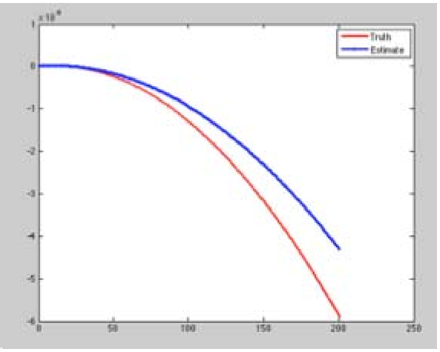
Final estimates compared with the truths for **UCB:**



AM Rush Hour

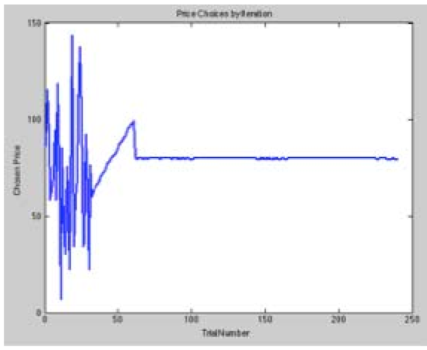


PM Rush Hour

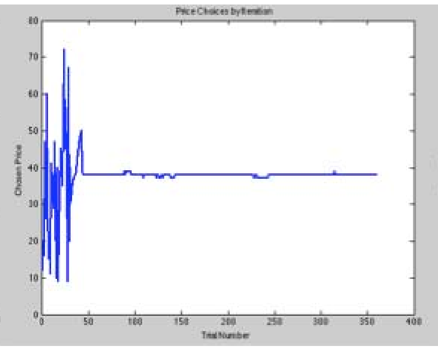


Graveyard Shift

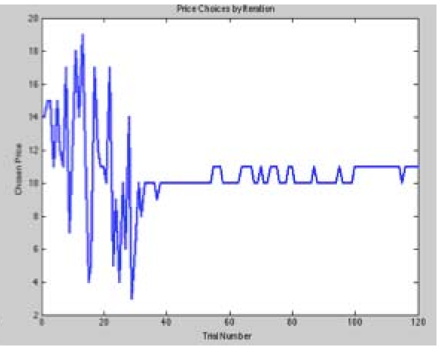
Prices chosen by Iteration, **UCB:**



AM Rush Hour



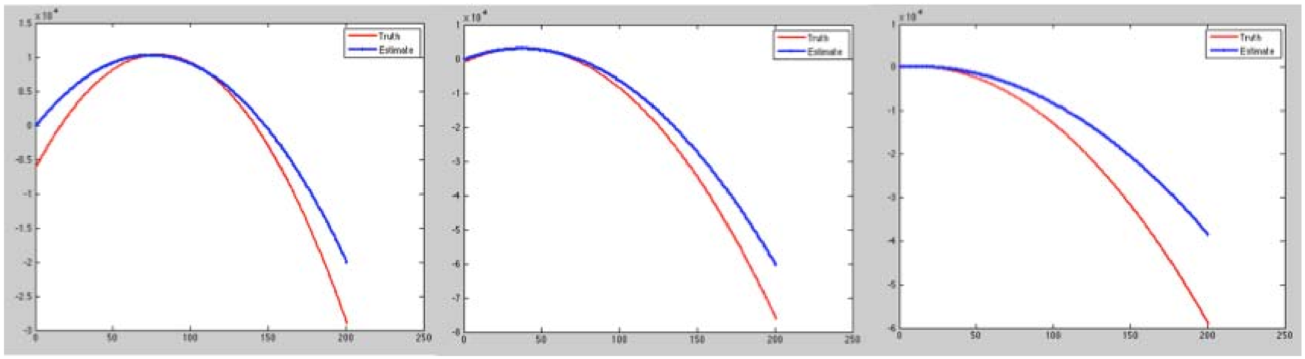
PM Rush Hour



Graveyard Shift

Online **Knowledge Gradient**, like exploitation, did not run into many of the problems we had earlier. Quite frankly, it was a little anti-climactic. It did well. Expected.

Final estimates compared with the truths for **KGCB, Online:**



AM Rush Hour PM Rush Hour Graveyard Shift

We scaled our truths for the lookup and linear models to be on the same scale and to be comparable. In general, our lookup table implementation worked a bit better, though the difference is pretty small. However, we felt much more confident in the lookup table since we made fewer constraints on our policies.

6. Conclusion

6.1 Summary

Table 6.1.1 – Lookup Table Belief Model

Lookup Table Belief Model [*** = not tested]									
	AM Rush Hour (7-9 AM)			PM Rush Hour (5-8 PM)			Graveyard Shift (2-4 AM)		
	Offline	Online		Offline	Online		Offline	Online	
Policy	Avg. Ω	Avg. Ω	Avg. ∑ Revenue	Avg. Ω	Avg. Ω	Avg. ∑ Revenue	Avg. Ω	Avg. Ω	Avg. ∑ Revenue
Exploit	798	798	$2.288M	215	215	$0.996M	**0.71**	**0.71**	**$12,796**
Explore	103	103	$1.252M	52	52	$0.566M	2.11	2.11	$4,743
IE	47	***	***	22	***	***	1.19	***	***
UCB	***	81	$2.298M	***	44	$1.016M	***	1.72	$10,610
KGCB	**45**	**72**	**$2.452M**	**19**	**40**	**$1.060M**	1.35	0.84	$12,579

Our results are consistent with prior literature that compares these policies.

Pure Exploitation as the baseline was consistently the worst-performing policy for AM Rush Hour and PM Rush Hour in offline settings. For Graveyard Shift, we see that Pure Exploitation performed the best; we do note that upon observation, our prior for the Graveyard Shift was an exceptionally close fit to the true revenue function; as such, Pure Exploitation was able to do particularly well. In the online setting, Pure Exploitation performed well, approaching UCB in average cumulative revenue accrued. This makes sense, as exploitation eschews uncertainty in favor of consistently high returns for each iteration.

In terms of average opportunity cost, we see that Pure Exploration did reasonably well, and significantly better than the baseline in all but Graveyard Shift in an offline setting. With correlated beliefs updating, exploration is able to amass large quantities of data that help fit the curve. It falls through in an online setting, however, consistently performing the worst in terms of cumulative revenue. This is to be expected, due to the policy's tendency to ignore any information about expected revenue.

Interval Estimation and UCB both performed very well, second only to the Knowledge Gradient. We do note that our tuning for z_α may not have been the most rigorous, and as such Interval Estimation has the potential to outperform Knowledge Gradient—in fact, it *did* outperform the Knowledge Gradient in the Graveyard Shift.

The Knowledge Gradient performed the best in all but one setting, the Graveyard Shift—as we expected.

Table 6.1.2 – Local Linear Approximation of Logistic Belief Model

Local Linear Approximation of Logistic Belief Model									
	AM Rush Hour (7-9 AM)			PM Rush Hour (5-8 PM)			Graveyard Shift (2-4 AM)		
	Offline	Online		Offline	Online		Offline	Online	
Policy	Avg. Ω	Avg. Ω	Avg. ∑ Revenue	Avg. Ω	Avg. Ω	Avg. ∑ Revenue	Avg. Ω	Avg. Ω	Avg. ∑ Revenue
Exploit	369	383	$2.384M	103	102	$1.032M	3.7	3.9	**$12,472**
Explore	**0.83**	**0.67**	$1.668M	42.8	41.3	$0.722M	**2.5**	2.6	$9,042
UCB	7.6	4.4	$2.365M	4.5	**9.0**	$1.042M	13.5	15.3	$11,806
KGCB	109	115	**$2.415M**	**2.8**	9.6	**$1.063M**	3.1	**2.4**	$12,374

Our results are consistent with prior literature that compares these policies.

Pure Exploitation as the baseline was consistently the worst-performing policy for AM Rush Hour and PM Rush Hour in offline settings. In the online setting, Pure Exploitation performed well, surpassing UCB in average cumulative revenue accrued for PM Rush Hour and Graveyard Shift. Again, this makes sense, as exploitation eschews uncertainty in favor of consistently high returns for each iteration.

In terms of average opportunity cost, we see that Constrained Exploration did extremely well, always outperforming baseline and earning the best average opportunity cost in AM Rush Hour and coming very close in the Graveyard Shift. Again, the strength of constrained exploration lies in the offline area, with the policy performing dismally with regards to average cumulative revenue but coming up with a very strong final approximation of the true quadratic curve.

UCB performed well, beating Knowledge Gradient cleanly for AM Rush Hour and coming fairly close in the PM Rush Hour. We see UCB performing rather poorly in the graveyard shift, which makes sense, as we allocate 30 of the budget to pure exploration to populate the estimation. In the Graveyard Shift, this amounts to a quarter of the total budget spent on pure exploration, which means that average cumulative revenue takes a hit.

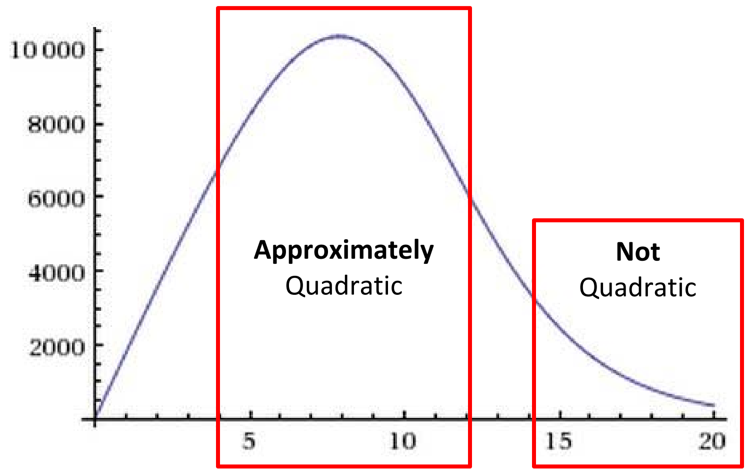
The Knowledge Gradient performed the best in every online setting, despite the issues outlined in **6.2 Major Limitations**.

6.2 Major Limitations

The major limitation of our belief model lies with the Local Linear Approximation of our Logistic Belief Model. We were not able to linearize our original true revenue function:

$$R(P) = \frac{P \cdot M}{1 + e^{-\mu_1 + \mu_2 P}}$$

on account of the price term in the numerator. As such, we were forced to use a linear model to approximate the portion of the logistic curve near the peak:



We also used a quadratic true revenue function, which was a quadratic regression on a set of points at the top of the logistic-based true revenue curve. With the quadratic truth function, certain sets of prices would return negative revenue as an observation. The problem we had with this was three-fold:

1. If we take our truth from the quadratic truth, we cannot justify it in context of reality. It is not realistic to observe negative revenue. Because the pay lanes have already been constructed and we assume no maintenance costs, the government must observe non-negative revenue during each sample period.

2. If we generate our truth using the logistic-based true revenue function from the Lookup Table belief model, then there is the possibility of choosing points past the point of inflection, where the true revenue function is concave upwards. In this range of points, the revenue function is certainly *not* quadratic. As a result, updating our quadratic estimate using these observed points will deform the estimated quadratic curve, most commonly by widening the curve.

3. If we take our truth from the quadratic truth but add a conditional statement that converted negative observations into zero, we run into the same issue as (2).

We tested all three methods of truth generation and ultimately decided to use the first method of truth generation (quadratic truth with no non-negativity constraint), as the other two methods led to particularly deformed graphs that lent high variability in the results for Exploration, UCB, and Knowledge Gradient policies. Thus, we chose to go with the first method to ensure that we could still compare policies.

We attempted to somewhat mitigate the problem by constraining our exploration, UCB, and Knowledge Gradient policies in the methods described in **3.2 Constrained Exploration** and **3.4 Modified Upper Confidence Bound.**

6.3 Possible Extensions

1. *Additional Variables* – As noted above, our problem contains several simplifications. One way to complicate the model is to account for different parameters. Instead of testing for three separate time periods, we could add a variable representing the hour of the day. We could also add binary variables representing rain, snow, or heat waves. Furthermore, we could account for a changing amount of total overall traffic rather than setting it equal to a constant.

2. *Separate Lanes* – We could relax our simplification regarding a single pay lane and a single regular lane. In this way we could model the effects of changing the number of lanes, or the sections of the road where two pay lanes merge into one pay lane. We could also model different prices for each pay lane, thus effectively changing the pay area of the road into a series of graded lanes with a gradient of shifting demands.

3. *Accounting for congestion* – Currently, we only take revenue as a response variable. However, we know that relieving congestion is also a priority of the government. Here, we are assuming that the drivers' utility from relieving/escaping congestion is contained within their demand function. However, we can also add congestion relief to the government's objection function:

$$\mu = R(P) + \int_{N-D(P)}^{N} \Psi(P)\, dp$$

   where $\Psi(P)$ is the marginal utility the government derives from reducing congestion in the regular lanes by one car, converted into monetary value. We integrate this from N-D(P) to N, representing the cars moving to the pay lane (D(P)) from the total overall traffic (N).

Appendix – Code

lookupExample.m – Information for the Lookup Table Belief Model

```
x=0.10*(0:200)'; %domain of the problem [prices]

peakAADT=21500*(2/3)/6;
%average number of cars total in sample period
% in our case, 10 minute interval during specific time of day (ie rush
% hour)

b = peakAADT/2; % for prior below

Pmax = 10; %gov't assumption that no one will pay more than Pmax

%Set the measurement variance (lambda) -- depends on AM,PM, or Grave
lambda=800^2;

mu_0=ones(201,1);

%our prior demand is linear from half of total traffic (because of two lanes)
at
% p=0 to no demand at p=20
% SO, our prior revenue is just our prior demand times p
for i=1:201
    if x(i)<=Pmax
       mu_0(i) = x(i)*(-b*(i-1)/(10*Pmax) + b);
    else
       mu_0(i) = 0;
    end
end

%generate the covariance matrix
    M=length(x);
    covM=zeros(M,M);

    %the matrix will be symmetric
    %instead of looping over all elements, just loop over the first half
    for i=1:M
        for j=(i+1):M
            covM(i,j)=lambda*exp(-0.3*abs(x(i)-x(j)));
        end
    end

    %and copy the rest to the other part
    covM=covM+covM';
    covM=covM+diag(lambda*ones(M,1));

mu1=3.7;
mu2=0.82;

%defined so that D(0) is equal to half of peak traffic
C = peakAADT/2*(1+exp(-mu1));
```

```
for i=1:M
    mu(i)= x(i)*(C/(1+exp(-mu1+mu2*x(i))));
end

%%
%THIS PART RUNS THE KGCB ON THE VALUES GENERATED BY ABOVE

beta_W=1/lambda*ones(M,1);
%Precision(Beta)=1/Variance

%Run the KGCB N many times
N=360;

%number of experiments
L=1;

totaloc=0;
revenuetotalavg=0;

for i=1:L
%The simple function to run KGCB
[mu_est, OC, choices, mu_estALL,revenuetotal]=lookup(mu,mu_0,beta_W,covM,N);
totaloc=totaloc+OC(N);
revenuetotalavg=revenuetotalavg + revenuetotal;
end

%disp(OC);
disp(totaloc/L);
disp(revenuetotalavg/L)


%Plot the final estimates vs. truth
figure;
plot(x,mu,'-r','LineWidth',2); %'-r' makes it a red line, LineWidth is the
thickness
hold on %hold the figure, as we'll also put on a second one
plot(x,mu_est,'.b','LineWidth',2); %'.b' makes it a blue dotted line
hold off
title('Final Estimates versus Truth');

%adjust the axes and add legends
%axis([1 100 min(mu)-0.5 max(mu)+0.5]);
legend('Truth','Estimate');

%Plot the estimate at the Kth time step vs. truth
K=1;
figure;
plot(x,mu,'-r','LineWidth',2); %'-r' makes it a red line, LineWidth is the
thickness
hold on %hold the figure, as we'll also put on a second one
plot(x,mu_0,'.b','LineWidth',2); %'.b' makes it a blue dotted line
%plot(x,mu_estALL(:,K),'.b','LineWidth',2); %'.b' makes it a blue dotted line
hold off
title('True Revenue and Prior');
```

```
%adjust the axes and add legends
%axis([1 100 min(mu)-0.5 max(mu)+0.5]);
legend('Truth','Estimate');

%Plot Opportunity Cost vs Time
figure;
plot(1:N,OC,'-g','LineWidth',1.5);
title('Opportunity Cost');
xlabel('Iteration');
ylabel('Opportunity Cost');

%or the log of it
%semilogy(1:N,OC,'-g','LineWidth',1.5);
%xlabel('Iteration');
%ylabel('Log_{10}(Opportunity Cost)');
%}
```

lookup.m – simulating the Lookup Table Belief Model

```
notation for the following:
K is the number of alternatives.
M is the number of time-steps
K x M stands for a matrix with K rows and M columns

This function takes in
mu:     true values for the mean (K x 1)
mu_0:   prior for the mean (K x 1)
beta_W: measurement precision (1/lambda(x)) (K x 1)
covM:   initial covariance matrix (K,K)
M:      how many measurements will be made (scalar)


And returns
mu_est:     Final estimates for the means (K x 1)

OC:         Opportunity cost at each iteration (1 x M)
choices:    Alternatives picked at each iteration (1 x M)
mu_estALL:  Estimates at each iteration (K x M)
%}

function [mu_est, OC, choices, mu_estALL,
revenuetotal]=lookup(mu,mu_0,beta_W,covM,M)

K=length(mu_0); %number of available choices
mu_est=mu_0;

OC=[];
choices=[];
revenueiter=[];
revenuecum=[];
revenuetotal=0;
mu_estALL=[];
count=zeros(K);
```

```
T=0; %only for UCB below
%{
%----------------------
%SAMPLE TO FIND RANGE TO LIMIT SEARCH OF UCB

T=30; %Number in UCB Exploration Sample

for i=1:T
    x = ceil(rand()*51);
    W_k=mu(x)+randn(1)*1./sqrt(beta_W(x));

    if W_k<0
        W_k=0;
    end

    e_x=zeros(K,1);
    e_x(x)=1;

    %updating equations for Normal-Normal model with covariance
    addscalar = (W_k - mu_est(x))/(1/beta_W(x) + covM(x,x));
    mu_est=mu_est + addscalar*covM*e_x;
    covM = covM - (covM*e_x*e_x'*covM)/((1/beta_W(x)) + covM(x,x));

    [max_est, max_choice]=max(mu_est);

    %calculate the opportunity cost
    o_cost=max(mu)-mu(max_choice);

    OC=[OC,o_cost]; %update the OC matrix
    choices=[choices, x]; %update the choice matrix

    %update revenue
    revenuetotal=revenuetotal+W_k;
    revenueiter=[revenueiter, W_k];
    revenuecum=[revenuecum, revenuetotal];


    count(x) = count(x)+1;

end
    uncertainty=std(revenueiter);
    r = max_est-.25*uncertainty;



    mu_esthalf1=mu_est(1:max_choice);

    mu_esthalf2=mu_est(max_choice:K);
    [~,a]=min(abs(mu_esthalf1-r));
    [~,b]=min(abs(mu_esthalf2-r));
    b=b+max_choice;
    disp(a);
    disp(b);
    disp(r);
```

```
%----------------------
%}
for k=1:(M-T) %try the kgcb for M number of times

    %Plogy is the log values of KG for alternatives
    Plogy=[];
    Py=[];
    onlinePy=[];

    %KNOWLEDGE GRADIENT
   %{
    for iter1=1:K
        a=mu_est';
        b=covM(iter1,:)/sqrt(1/beta_W(iter1) + covM(iter1,iter1));

        [KG,LogKG] = LogEmaxAffine(a,b);

        [Plogy]=[Plogy, LogKG];
        [Py]=[Py,KG];

    end
    %[maxh,x]=max(Plogy);


    %ONLINE KG

    for i=1:K
        onlinevalue = mu_est(i) + (M-k)*Py(i);
        [onlinePy] = [onlinePy, onlinevalue];
    end

    [maxh,x]=max(onlinePy);
   %}

   %PURE EXPLORATION
   x = ceil(rand()*101);

   %PURE EXPLOITATION
   % [maxh,x]=max(mu_est);

   %INTERVAL ESTIMATION
  %{
   zalpha = 3;
   sigma=ones(K,1);
   v=ones(K,1);
   for i=1:K

        sigma(i) = sqrt(covM(i,i));

        v(i) = mu_est(i) + zalpha*sigma(i);
   end

    [best_value, x] = max(v);
```

```
%}

 %UPPER CONFIDENCE BOUNDING FOR ONLINE
%{
 v=zeros(K,1);
 for i=a:b
     %v(i) = mu_est(i) + 4*(1/beta_W(i))*sqrt(log(k)/(count(i)));
     v(i) = mu_est(i) + sqrt(2*log(k)/(count(i)));
 end

 [best_value, x] = max(v);

 %}

 %max_value is the best estimated value of the KG
 %x is the argument that produces max_value

 %observe the outcome of the decision
 %W_k=mu_k+Z*SigmaW_k where SigmaW is standard deviation of the
 %error for each observation
 W_k=mu(x)+randn(1)*1./sqrt(beta_W(x));

 if W_k<0
     W_k=0;
 end

 e_x=zeros(K,1);
 e_x(x)=1;

 %updating equations for Normal-Normal model with covariance
 addscalar = (W_k - mu_est(x))/(1/beta_W(x) + covM(x,x));
 mu_est=mu_est + addscalar*covM*e_x;
 covM = covM - (covM*e_x*e_x'*covM)/((1/beta_W(x)) + covM(x,x));

 %pick the best one to compare OC
 [max_est, max_choice]=max(mu_est);

 %calculate the opportunity cost
 o_cost=max(mu)-mu(max_choice);

 OC=[OC,o_cost]; %update the OC matrix
 choices=[choices, x]; %update the choice matrix

 %update revenue
 revenuetotal=revenuetotal+W_k;
 revenueiter=[revenueiter, W_k];
 revenuecum=[revenuecum, revenuetotal];


 count(x) = count(x)+1;

 if nargout>3 %if more than three outputs were asked
     mu_estALL=[mu_estALL,mu_est];
 end
```

```
end


figure;
bar(count, 'LineWidth',2);
title('Prices Chosen by Frequency');
xlabel('Price');
ylabel('Frequency');
figure;
plot(choices, 'LineWidth',2);
title('Price Choices by Iteration');
xlabel('Trial Number');
ylabel('Chosen Price');

figure;
plot(revenueiter, 'LineWidth',2);
title('Revenue by Iteration');
xlabel('Trial Number');
ylabel('Revenue');
figure;
plot(revenuecum, 'LineWidth',2);
title('Cumulative Revenue');
xlabel('Trial Number');
ylabel('Revenue');
%}
end




% logy = LogEmaxAffine(a,b)
% Calculates log(Exp[max_x a_x + b_x Z]-max_x a_x), where Z is a standard
% normal random variable and a,b are 1xM input vectors.
function [y,logy, a,b,c] = LogEmaxAffine(a,b)
    if (any(isnan(a)) || any(isnan(b)))
        warning('a or b is NaN');
    end
    assert(all(isreal(a)));
    assert(all(isreal(b)));

    a = a';
    b = b';

    % Check that a and b are column vectors of the right size
    if (any(size(a) ~= size(b)))
        error('LogEmaxAffine: a and b must be column vectors of the same
size');
    end

    [a,b] = AffineBreakpointsPrep(a,b);

    [c, keep] = AffineBreakpoints(a,b);
    a = a(keep);
    b = b(keep);
    c = c([1,keep+1]);
    M = length(keep);
    assert(all(isreal(c)));

    % I need logbdiff=log(diff(b)).  I thought that the following code would
be
    % more numerically stable, able for example to distinguish cases like
    % logb = [-25 -.3] vs. logb = [-35 -.3], but it doesn't seem to be able
to.
    % Indeed, in the debugging output that I have below, the difference was 0.
    %{
    logb = log(abs(b)); % If b is 0, this is -Inf.
    sgnb = sign(b); % If b is 0, this is 0.
    logbdiff = zeros(size(c(2:M)));
    for i=1:length(b)-1
    [logbdiff(i),logbdiffsgn] = LogPlusExpSigned(logb(i),sgnb(i),logb(i+1),-
sgnb(i+1));
    %assert(logbdiffsgn>=0);  % The b are distinct, so bdiff(i) can't be 0.
    end
    disp(sprintf('log(b)=%s log(diff(b))=%g logbdiff=%g
difference=%g',mat2str(log(b)),log(diff(b)),logbdiff,log(diff(b))-logbdiff));
    %}
    logbdiff = log(diff(b))';

    if M==1
        logy=log(a);
    elseif M>=2
        logy = LogSumExp(logbdiff+LogEI(-abs(c(2:M))));
    end

    logy=real(logy);
    y=exp(logy);
end
% Prepares vectors for passing to AffineEmaxBreakpoints, changing their
% order and removing elements with duplicate slope.

function [a,b] = AffineBreakpointsPrep(a,b)
    % Make sure a and b are column vectors.
    rows = size(a); if (rows == 1), a=a'; end
    rows = size(b); if (rows == 1), b=b'; end

    % 11/29/2008 PF: Experimental preprocessing step, which I hope will
remove
    % a large number of the entries.
    [b1, i1] = min(b); % [a1,b1] is best at z=-infinity
    [a2, i2] = max(a); % [a2,b2] is best at z=0
    [b3, i3] = max(b); % [a3,b3] is best at z=+infinity
    a1 = a(i1);
    b2 = b(i2);
    a3 = a(i3);
    cleft = (a - a1)./(b1 - b); % intersection with leftmost line.
    cright = (a - a3)./(b3 - b); % intersection with rightmost line.
    c2left = (a2 - a1)./(b1 - b2); % intersection with leftmost line.
    c2right = (a2 - a3)./(b3 - b2); % intersection with rightmost line.
    keep = find(b==b1 | b==b3 | cleft <= c2left | cright >= c2right);
    %disp(sprintf('Preprocessing cut %d of %d entries', length(a)-
length(keep), length(a)));
    a = a(keep);
    b = b(keep);
    clear keep cleft cright



    % Form a matrix for which ba(x,1) is the slope b(x) and ba(x,2) is the
    % y-intercept a(x).  Sort this matrix in ascending order of slope,
    % breaking ties in slope with the y-intercept.
    ba = [b, a];
    ba = sortrows(ba,[1,2]);
    a = ba(:,2);
    b = ba(:,1);

    % Then, from each pair of indices with the b component equal, remove
    % the one with smaller a component.  This code works because the sort
    % above enforced the condition: if b(i) == b(i+1), then a(i) <= a(i+1).
    keep = [find(diff(b)); length(b)];
    % This previous line is equivalent to:
    % keep = [];
    % for i=[1:length(b)-1]
    %    if b(i)~=b(i+1)
    %        keep = [keep, i];
    %    end
    %end
    %keep = [keep, length(b)];  % We always keep the last one.

    % Note that the elements of keep are in ascending order.
    % This makes it so that b(keep) is still sorted in ascending order.
    a = a(keep);
    b = b(keep);
end

% Inputs are two M-vectors, a and b.
% Requires that the b vector is sorted in increasing order.
% Also requires that the elements of b all be unique.
% This function is used in AffineEmax, and the preparation of generic
% vectors a and b to satisfy the input requirements of this function are
% shown there.
%
% The output is an (M+1)-vector c and a vector A ("A" is for accept).  Think
of
% A as a set which is a subset of {1,...,M}.  This output has the property
% that, for any i in {1,...,M} and any real number z,
%   i \in argmax_j a_j + b_j z
% iff
%   i \in A and z \in [c(j+1),c(i+1)],
%   where j = sup {0,1,...,i-1} \cap A.
%
% A note about indexing:
% Since Matlab does not allow indexing from 0, but instead requires
% indexing from 1, what is called c_i in the paper is written in matlab as
% c(1+i).  This is because in the paper we reference c_0.  For the vectors
% a and b, however, we don't need to reference a_0 or b_0, so we reference
% a_i and b_i by a(i) and b(i) respectively, rather than a(i+1) or b(i+1).
%
function [c,A] = AffineBreakpoints(a,b)
    % Preallocate for speed.  Instead of resizing the array A whenever we add
    % to it or delete from it, we keep it the maximal size, and keep a length
    % indicator Alen telling us how many of its entries are good.  When the
    % function ends, we remove the unused elements from A before passing
    % it.
    M = length(a);
    c = zeros(1,M+1);
    A = zeros(1,M);

    % Step 0
    i=0;
    c(1+i) = -inf;
    c(1+i+1) = +inf;
    A(1) = 1;
    Alen = 1;

    for i=[1:M-1]
        c(1+i+1) = +inf;
        while(1)
            j = A(Alen); % jindex = Alen
            c(1+j) = (a(j) - a(i+1))/(b(i+1)-b(j));
        % The if statement below replaces these lines from version 2 of the
        % function.
        %    kindex = jindex-1 = Alen-1
            %    if kindex > 0 && c(1+j)<=c(1+A(kindex))
            if Alen > 1 && c(1+j)<c(1+A(Alen-1))
        Alen = Alen-1; % Remove last element j
                % continue in while(1) loop
            else
                break % quit while(1) loop
            end
        end
    A(Alen+1) = i+1;
    Alen = Alen + 1;
    end
    A = A(1:Alen);
end

% Returns the log of Exp[(s+Z)^+], where s is a constant and Z is a standard
% normal random variable.  For large negative arguments Exp[(s+Z)^+] function
% is close to 0.  For large positive arguments, the function is close to the
% argument.  For s large enough, s>-10, we use the formula
% Exp[(s+Z)^+] = s*normcdf(s) + normpdf(s).  For smaller s we use an
asymptotic
% approximation based on Mill's ratio.  EI stands for "expected improvement",
% since Exp[(s+Z)^+] would be the log of the expected improvement by
measuring
% an alternative with excess predictive mean s over the best other measured
% alternative, and predictive variance 0.
function logy = LogEI(s)

% Use the asymptotic approximation for these large negative s.  The
% approximation is derived via:
```

```
%   s*normcdf(s) + normpdf(s) = normpdf(s)*[1-|s|normcdf(-|s|)/normpdf(s)]
% and noting that normcdf(-|s|)/normpdf(s) is the Mill's ratio at |s|, which
is
% asymptotically approximated by |s|/(s^2+1) [Gordon 1941, also documented in
% Frazier,Powell,Dayanik 2009 on page 14].  This gives,
%   s*normcdf(s) + normpdf(s) = normpdf(s)*[1-s^2/(s^2+1)] =
normpdf(s)/(s^2+1).

i=find(s<-10);
if (length(i)>0)
    logy(i) = LogNormPDF(s(i)) - log(s(i).^2 + 1);
end

% Use straightforward routines for s in the more numerically stable region.
i=find(s>=-10);
if (length(i)>0)
    logy(i) = log(s(i).*normcdf(s(i))+normpdf(s(i)));
end

assert(all(isreal(logy)));
end

% logy = LogNormPDF(z)
% Returns the log of the normal pdf evaluated at z.  z can be a vector or a
scalar.
function logy = LogNormPDF(z)
    const = -.5*log(2*pi); % log of 1/sqrt(2pi).
    logy = const - z.^2/2;
end

% function y=LogSumExp(x)
% Computes log(sum(exp(x))) for a vector x, but in a numerically careful way.
function y=LogSumExp(x)
xmax = max(x);
y = xmax + log(sum(exp(x-xmax)));
end
```

quadraticExample.m – Information for the Linear Belief Model

```
%%
%THIS PART GENERATES THE TRUTH
%X (Regression Matrix)
%theta_0 (PRIOR BELIEF ABOUT THE REGRESSION VALUES)
%theta (TRUE VALUES FOR THE REGRESSION)
%truth (TRUE VALUES FOR PARAMETERS)
%(in a unreal perfect setting where this model explains everything
truth=X*theta)

%offline = 1 means run offline KG; offline = 0 means run online
offline = 0;


%First form the X Matrix
```

```
%price
x_i1=0.10*(0:200)';

x_i2=ones(201,1);

%price squared
for i=1:201
    x_i2(i)= (x_i1(i))^2;
end

%X=[1,p,p^2]
X=[ones(201,1),x_i1,x_i2];

disp(X);

%Prior Estimates
theta_0=[-93.75;431.25;-250];

%Pick a true value for theta
theta=[-33.19;302.82;-162.99];

%Set the prior on covariance matrix, assume independence
C=zeros(3,3);
C(1,1)=44322.91667;
C(1,2)=-82552.08333;
C(1,3)=36666.66667;
C(2,1)=-82552.08333;
C(2,2)=158072.9167;
C(2,3)= -74166.66667;
C(3,1)=36666.66667;
C(3,2)= -74166.66667;
C(3,3)=38333.33333;

%Set the measurement variance
MVar=40^2;

%Set the true values as X*theta
truth=X*theta;

disp(truth);

disp(C);
%%
%THIS PART RUNS THE KGCBLinR ON THE VALUES GENERATED BY ABOVE

%Run the KGCBLinR L many times
L=120;

totaloc=0;
revenuetotalavg=0;

%number of experiments
N=100;
```

```
for i=1:N
    [thetaLast,OC,choices,thetaEst,
revenuetotal]=quadratic(X,theta_0,MVar,truth,L,C,offline,1);
    totaloc=totaloc+OC(L);
    revenuetotalavg=revenuetotalavg + revenuetotal;
end

disp('Ended');
disp(totaloc/N);
disp(revenuetotalavg/N);



%{
%Plot the final estimates vs. truth
figure;
plot(truth,'-r','LineWidth',2); %'-r' makes it a red line, LineWidth is the
thickness
hold on %hold the figure, as we'll also put on a second one
plot(X*thetaLast,'.-b','LineWidth',2); %'.b' makes it a blue dotted line
hold off

%add legends
legend('Truth','Estimate');

figure;
plot(truth,'-r','LineWidth',2); %'-r' makes it a red line, LineWidth is the
thickness
hold on %hold the figure, as we'll also put on a second one
plot(X*theta_0,'.-b','LineWidth',2); %'.-b' makes it a blue dotted line
hold off

%add legends
legend('Truth','Estimate');

%Plot Opportunity Cost vs Time
figure;
plot(1:L,OC,'-g','LineWidth',1.5);
xlabel('Iteration');
ylabel('Opportunity Cost');

%}
```

quadratic.m – simulating the Linear Belief Model

```
%{
notation for the following:
K is the number of features (variables in the model).
M is the number of alternatives.  This may be quite large.
N is the number of alternatives we have actually sampled.
K x M stands for a matrix with K rows and M columns

This function takes in
X:      the design matrix for the linear regression model (K x N)
theta_0:prior for the linear regression parameters (N x 1)
MVar:   variance of measurement noise (scalar)
```

```
truth:  true values for the means of alternatives (K x 1)
L:      how many measurements will be made (scalar)

C: Prior covariance matrix for theta. If nothing is used, the default
parameter is
(X'*X)^-1*MVar (the empirical estimate)

Weigher: This is a scalar between 0 and 1 (default is 1).
If the mesaurements are from a non-stationary distribution, a smaller
"Weigher" puts less weight on the previous observations.
Please see pg. 145 of the book (and the paragraph before equation 7.7).

And returns
theta:      Final estimates for the lin. reg. parameters (N x 1)

OC:         Opportunity cost at each iteration (1 x M)
choices:    Alternatives picked at each iteration (1 x M)

thetaEst:  Estimates at each iteration (N x M)
%}

function
[theta,OC,choices,thetaEst,revenuetotal]=quadratic(X,theta_0,MVar,truth,L,C,o
ffline,Weigher)

theta=theta_0;

[rows,n] = size(X);

OC=[];
choices=[];
thetaEst=[];
revenueiter=[];
revenuecum=[];
revenuetotal=0;
count=zeros(rows);
T=0;

%{
%-----------------------
%SAMPLE TO FIND RANGE TO LIMIT SEARCH OF UCB

T=30; %Number in UCB Exploration Sample

for i=1:T
    currentestimates = X*theta;
    for i=2:rows
        if (currentestimates(i)>=0 && currentestimates(i-1)<0)
            a=i;
        end
        if (currentestimates(i)<=0 && currentestimates(i-1)>0)
            b=(i-1);
        end
    end
    choice = a + ceil(rand()*(b-a));

    %Observe the choice
    observation=truth(choice)+randn(1)*sqrt(MVar);

    if observation<0
        observation=0;
    end

    %Few things for the recursive updating eq.
    errorU=observation-theta'*X(choice,:)';
    gammaU=Weigher*(MVar)+X(choice,:)*C*X(choice,:)';

    %Update parameters
    theta=theta+(errorU./gammaU)*C*X(choice,:)';
    C=1./Weigher*(C-(1./gammaU)*C*X(choice,:)'*X(choice,:)*C);

    %pick the best one to compute OC
    [max_est, max_choice]=max(X*theta);

    %calculate the opportunity cost
    o_cost=max(truth)-truth(max_choice);

    OC=[OC,o_cost]; %update the OC matrix
    choices=[choices, choice]; %update the choice matrix

    %update revenue
    revenuetotal=revenuetotal+observation;
    revenueiter=[revenueiter, observation];
    revenuecum=[revenuecum, revenuetotal];


    count(choice) = count(choice)+1;

end
    uncertainty=std(revenueiter);
    r = max_est-.25*uncertainty;

    mu_est = X*theta;

    mu_esthalf1=mu_est(1:max_choice);

    mu_esthalf2=mu_est(max_choice:rows);
    [~,a]=min(abs(mu_esthalf1-r));
    [~,b]=min(abs(mu_esthalf2-r));
    b=b+max_choice-1;
    disp(a);
    disp(b);
    disp(r);

%----------------------
%}
```

```
for n=1:(L-T)

    %Choose using KG
    %choice=KGCBLin(theta,C,X,MVar,n,L,offline);

    %{
    %PURE EXPLORATION
    %
    currentestimates = X*theta;
    for i=2:rows
        if (currentestimates(i)>=0 && currentestimates(i-1)<0)
            a=i;
        end
        if (currentestimates(i)<=0 && currentestimates(i-1)>0)
            b=(i-1);
        end
    end

    choice = a + ceil(rand()*(b-a));
    %}


    %PURE EXPLOITATION
    currentestimates = X*theta;
    [maxrevest,choice]=max(currentestimates);
    %}
    %{
    %UPPER CONFIDENCE BOUNDING FOR ONLINE
    mu_est = X*theta;

    v=zeros(rows,1);
    for i=a:b
        v(i) = mu_est(i) + sqrt(2*log(n)/(count(i)));
    end

    [best_value, choice] = max(v);

    %}

    %Observe the choice
    observation=truth(choice)+randn(1)*sqrt(MVar);


    if observation<0
        observation=0;
    end

    %Few things for the recursive updating eq.
    errorU=observation-theta'*X(choice,:)';
    gammaU=Weigher*(MVar)+X(choice,:)*C*X(choice,:)';

    %Update parameters
    theta=theta+(errorU./gammaU)*C*X(choice,:)';
    C=1./Weigher*(C-(1./gammaU)*C*X(choice,:)'*X(choice,:)*C);

    %pick the best one to compute OC
    [max_est, max_choice]=max(X*theta);

    %calculate the opportunity cost
    o_cost=max(truth)-truth(max_choice);

    OC=[OC,o_cost]; %update the OC matrix
    choices=[choices, choice]; %update the choice matrix

    %update revenue
    revenuetotal=revenuetotal+observation;
    revenueiter=[revenueiter, observation];
    revenuecum=[revenuecum, revenuetotal];

    count(choice) = count(choice)+1;


    if nargout>3
        thetaEst=[thetaEst,theta];
    end

end
%{
figure;
bar(count, 'LineWidth',2);
title('Prices Chosen by Frequency');
xlabel('Price');
ylabel('Frequency');
figure;
plot(choices, 'LineWidth',2);
title('Price Choices by Iteration');
xlabel('Trial Number');
ylabel('Chosen Price');

figure;
plot(revenueiter, 'LineWidth',2);
title('Revenue by Iteration');
xlabel('Trial Number');
ylabel('Revenue');
figure;
plot(revenuecum, 'LineWidth',2);
title('Cumulative Revenue');
xlabel('Trial Number');
ylabel('Revenue');
%}

end

%KGCBLin uses the linear model to choose what appears to be the best
%alternative.

%X matrix for alternatives (M,K)
```

```
%Theta belief about parameters (K,1)
%C covariance for Theta (K,K)
%MVar is a scalar for measurement variance

function [bestX,KGVal,maxKG]=KGCBLin(theta,C,X,MVar,n,L,offline)

[rows, N]=size(X);
mu=X*theta;

a=1;
b=201;

for i=2:rows
        if (mu(i)>=0 && mu(i-1)<0)
            a=i;
        end
        if (mu(i)<=0 && mu(i-1)>0)
            b=(i-1);
        end
    end

B=X*C;

offline_KGVal=[];
online_KGVal=[];
KGVal=[];

for i=a:b
    Sigma=B*(X(i,:)');
    b=Sigma./sqrt(MVar+Sigma(i));
    [KG,LogKG] = LogEmaxAffine(mu',b');
    offline_KGVal=[offline_KGVal; KG];
    online_KGVal=[online_KGVal; mu(i)+(L-n)*KG];
    if offline==0
        KGVal=online_KGVal;
    elseif offline>0
        KGVal=offline_KGVal;
    end
end

%disp(KGVal);

[maxKG, bestX]=max(KGVal);

end

% logy = LogEmaxAffine(a,b)
% Calculates log(Exp[max_x a_x + b_x Z]-max_x a_x), where Z is a standard
% normal random variable and a,b are 1xM input vectors.
function [y,logy, a,b,c] = LogEmaxAffine(a,b)
    if (any(isnan(a)) || any(isnan(b)))
        warning('a or b is NaN');
    end
    assert(all(isreal(a)));
    assert(all(isreal(b)));
    a = a';
    b = b';

    % Check that a and b are column vectors of the right size
    if (any(size(a) ~= size(b)))
        error('LogEmaxAffine: a and b must be column vectors of the same
size');
    end

    [a,b] = AffineBreakpointsPrep(a,b);

    [c, keep] = AffineBreakpoints(a,b);
    a = a(keep);
    b = b(keep);
    c = c([1,keep+1]);
    M = length(keep);
    assert(all(isreal(c)));

    % I need logbdiff=log(diff(b)).  I thought that the following code would
be
    % more numerically stable, able for example to distinguish cases like
    % logb = [-25 -.3] vs. logb = [-35 -.3], but it doesn't seem to be able
to.
    % Indeed, in the debugging output that I have below, the difference was 0.
    %{
    logb = log(abs(b)); % If b is 0, this is -Inf.
    sgnb = sign(b); % If b is 0, this is 0.
    logbdiff = zeros(size(c(2:M)));
    for i=1:length(b)-1
    [logbdiff(i),logbdiffsgn] = LogPlusExpSigned(logb(i),sgnb(i),logb(i+1),-
sgnb(i+1));
    %assert(logbdiffsgn>=0);  % The b are distinct, so bdiff(i) can't be 0.
    end
    disp(sprintf('log(b)=%s log(diff(b))=%g logbdiff=%g
difference=%g',mat2str(log(b)),log(diff(b)),logbdiff,log(diff(b))-logbdiff));
    %}
    logbdiff = log(diff(b))';

    if M==1
        logy=log(a);
    elseif M>=2
        logy = LogSumExp(logbdiff+LogEI(-abs(c(2:M))));
    end

    logy=real(logy);
    y=exp(logy);
end
% Prepares vectors for passing to AffineEmaxBreakpoints, changing their
% order and removing elements with duplicate slope.

function [a,b] = AffineBreakpointsPrep(a,b)
    % Make sure a and b are column vectors.
    rows = size(a); if (rows == 1), a=a'; end
    rows = size(b); if (rows == 1), b=b'; end
    % 11/29/2008 PF: Experimental preprocessing step, which I hope will
remove
    % a large number of the entries.
    [b1, i1] = min(b); % [a1,b1] is best at z=-infinity
    [a2, i2] = max(a); % [a2,b2] is best at z=0
    [b3, i3] = max(b); % [a3,b3] is best at z=+infinity
    a1 = a(i1);
    b2 = b(i2);
    a3 = a(i3);
    cleft = (a - a1)./(b1 - b); % intersection with leftmost line.
    cright = (a - a3)./(b3 - b); % intersection with rightmost line.
    c2left = (a2 - a1)./(b1 - b2); % intersection with leftmost line.
    c2right = (a2 - a3)./(b3 - b2); % intersection with rightmost line.
    keep = find(b==b1 | b==b3 | cleft <= c2left | cright >= c2right);
    %disp(sprintf('Preprocessing cut %d of %d entries', length(a)-
length(keep), length(a)));
    a = a(keep);
    b = b(keep);
    clear keep cleft cright

    % Form a matrix for which ba(x,1) is the slope b(x) and ba(x,2) is the
    % y-intercept a(x).  Sort this matrix in ascending order of slope,
    % breaking ties in slope with the y-intercept.
    ba = [b, a];
    ba = sortrows(ba,[1,2]);
    a = ba(:,2);
    b = ba(:,1);

    % Then, from each pair of indices with the b component equal, remove
    % the one with smaller a component.  This code works because the sort
    % above enforced the condition: if b(i) == b(i+1), then a(i) <= a(i+1).
    keep = [find(diff(b)); length(b)];
    % This previous line is equivalent to:
    % keep = [];
    % for i=[1:length(b)-1]
    %    if b(i)~=b(i+1)
    %        keep = [keep, i];
    %    end
    %end
    %keep = [keep, length(b)];  % We always keep the last one.

    % Note that the elements of keep are in ascending order.
    % This makes it so that b(keep) is still sorted in ascending order.
    a = a(keep);
    b = b(keep);
end

% Inputs are two M-vectors, a and b.
% Requires that the b vector is sorted in increasing order.
% Also requires that the elements of b all be unique.
% This function is used in AffineEmax, and the preparation of generic
% vectors a and b to satisfy the input requirements of this function are
% shown there.
%
```

```
% The output is an (M+1)-vector c and a vector A ("A" is for accept).  Think
of
% A as a set which is a subset of {1,...,M}.  This output has the property
% that, for any i in {1,...,M} and any real number z,
%   i \in argmax_j a_j + b_j z
% iff
%   i \in A and z \in [c(j+1),c(i+1)],
%   where j = sup {0,1,...,i-1} \cap A.
%
% A note about indexing:
% Since Matlab does not allow indexing from 0, but instead requires
% indexing from 1, what is called c_i in the paper is written in matlab as
% c(1+i).  This is because in the paper we reference c_0.  For the vectors
% a and b, however, we don't need to reference a_0 or b_0, so we reference
% a_i and b_i by a(i) and b(i) respectively, rather than a(i+1) or b(i+1).
%
function [c,A] = AffineBreakpoints(a,b)
    % Preallocate for speed.  Instead of resizing the array A whenever we add
    % to it or delete from it, we keep it the maximal size, and keep a length
    % indicator Alen telling us how many of its entries are good.  When the
    % function ends, we remove the unused elements from A before passing
    % it.
    M = length(a);
    c = zeros(1,M+1);
    A = zeros(1,M);

    % Step 0
    i=0;
    c(1+i) = -inf;
    c(1+i+1) = +inf;
    A(1) = 1;
    Alen = 1;

    for i=[1:M-1]
        c(1+i+1) = +inf;
        while(1)
            j = A(Alen); % jindex = Alen
            c(1+j) = (a(j) - a(i+1))/(b(i+1)-b(j));
        % The if statement below replaces these lines from version 2 of the
        % function.
        %    kindex = jindex-1 = Alen-1
            %    if kindex > 0 && c(1+j)<=c(1+A(kindex))
            if Alen > 1 && c(1+j)<c(1+A(Alen-1))
        Alen = Alen-1; % Remove last element j
                % continue in while(1) loop
            else
                break % quit while(1) loop
            end
        end
    A(Alen+1) = i+1;
    Alen = Alen + 1;
    end
    A = A(1:Alen);
end

% Returns the log of Exp[(s+Z)^+], where s is a constant and Z is a standard
% normal random variable.  For large negative arguments Exp[(s+Z)^+] function
% is close to 0.  For large positive arguments, the function is close to the
% argument.  For s large enough, s>-10, we use the formula
% Exp[(s+Z)^+] = s*normcdf(s) + normpdf(s).  For smaller s we use an
asymptotic
% approximation based on Mill's ratio.  EI stands for "expected improvement",
% since Exp[(s+Z)^+] would be the log of the expected improvement by
measuring
% an alternative with excess predictive mean s over the best other measured
% alternative, and predictive variance 0.
function logy = LogEI(s)

% Use the asymptotic approximation for these large negative s.  The
% approximation is derived via:
%   s*normcdf(s) + normpdf(s) = normpdf(s)*[1-|s|normcdf(-|s|)/normpdf(s)]
% and noting that normcdf(-|s|)/normpdf(s) is the Mill's ratio at |s|, which
is
% asymptotically approximated by |s|/(s^2+1) [Gordon 1941, also documented in
% Frazier,Powell,Dayanik 2009 on page 14].  This gives,
%   s*normcdf(s) + normpdf(s) = normpdf(s)*[1-s^2/(s^2+1)] =
normpdf(s)/(s^2+1).

i=find(s<-10);
if (length(i)>0)
    logy(i) = LogNormPDF(s(i)) - log(s(i).^2 + 1);
end

% Use straightforward routines for s in the more numerically stable region.
i=find(s>=-10);
if (length(i)>0)
    logy(i) = log(s(i).*normcdf(s(i))+normpdf(s(i)));
end

assert(all(isreal(logy)));
end

% logy = LogNormPDF(z)
% Returns the log of the normal pdf evaluated at z.  z can be a vector or a
scalar.
function logy = LogNormPDF(z)
    const = -.5*log(2*pi); % log of 1/sqrt(2pi).
    logy = const - z.^2/2;
end

% function y=LogSumExp(x)
% Computes log(sum(exp(x))) for a vector x, but in a numerically careful way.
function y=LogSumExp(x)
xmax = max(x);
y = xmax + log(sum(exp(x-xmax)));
end
```

This paper represents our own work in accordance with University regulations.

SHUYANG LI

MAX KAPLAN